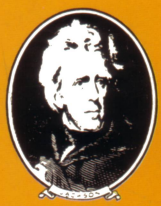
Z 80

PROGRAMMAZIONE IN LINGUAGGIO ASSEMBLY

EDIZIONE ITALIANA

LANCE A. LEVENTHAL

GRUPPO EDITORIALE JACKSON



Z 80

PROGRAMMAZIONE IN LINGUAGGIO ASSEMBLY

di
Lance
A. Leventhal

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

© Copyright per l'edizione originale Osborne/McGraw-Hill, Inc. 1979
© Copyright per l'edizione italiana Osborne/McGraw-Hill, Inc. 1981

Il Gruppo Editoriale Jackson ringrazia per il prezioso lavoro svolto nella stesura dell'edizione italiana le signore Francesca di Fiore, Rosi Bozzolo e l'Ing. Sergio Zannoli.
Traduzione a cura di Eds electronic data service - Bresso (Mi).

Tutti i diritti sono riservati. Stampato in Italia. Nessuna parte di questo libro può essere riprodotta, memorizzata in sistemi di archivio, o trasmessa in qualsiasi forma o mezzo, elettronico, meccanico, fotocopia, registrazione o altri senza la preventiva autorizzazione scritta dell'editore.

Fotocomposizione: Corponove s.n.c. — Bergamo
Stampa: Tipo Lito Ferrari Cesare & C. — Clusone (BG)

Questo libro è dedicato ai miei colleghi della Society for Computer Simulation - Romeo Favreau, Natalie Fowler, Alexander McKenna, John McLeod, Stanley Rogers, e Chip Stockton.

RINGRAZIAMENTI

L'autore desidera esprimere un ringraziamento alle seguenti persone:

Mr. Curt Ingraham, Ms. Mary Borchers e Ms. Janice Enger della Osborne and Associates, che fecero molte correzioni e diedero molti suggerimenti; Mr. Winthrop Saville di Sorrento Valley Associates, che diede assistenza e fece gli esempi; Mr. Tom Littlefield di Littlefield/Smith Associates, che fornì il materiale per i riferimenti bibliografici; Ms. Marielle Carter di Sorrento Valley Associates che scrisse a macchina parte del materiale; Mr. Stanley della Rogers della Society for Computer Simulation, che ha continuato a suggerire molti miglioramenti nello stile di scrittura dell'autore; sua moglie Donna, per la sua pazienza e la sua comprensione durante la scrittura di questo libro.

Altre persone che diedero assistenza e suggerimenti sono: Mr. Colin Walsh, Mr. Gary Hankins, Mr. Romeo Favreau, Mr. David Bulman, Mr. Kati Bulman, Mr. Robert Turner, Mr. Irv Stafford, Mr. John Burgar, Mr. Ferenc Montvai - Lako e Mr. Warren McKenna. Altri studenti e colleghi hanno aiutato l'autore sulla giusta traccia.

L'autore naturalmente declina responsabilità per ogni eventuale errore restante, concetti ed interpretazioni erronei.

INDICE

CAPITOLO		PAGINA
1	INTRODUZIONE ALLA PROGRAMMAZIONE IN LINGUAGGIO ASSEMBLY	1-1
	COME È STATO STAMPATO QUESTO LIBRO	1-1
	IL SIGNIFICATO DELLE ISTRUZIONI	1-1
	UN PROGRAMMA DI COMPUTER	1-2
	IL PROBLEMA DELLA PROGRAMMAZIONE	1-2
	USO DELL'OTTALE O DELL'ESADECIMALE	1-3
	MNEMONICA DEI CODICI D'ISTRUZIONE	1-5
	IL PROGRAMMA ASSEMBLATORE	1-5
	CARATTERISTICHE ADDIZIONALI DEGLI ASSEMBLATORI	1-6
	SVANTAGGI DEL LINGUAGGIO ASSEMBLY	1-7
	LINGUAGGI AD ALTO LIVELLO	1-7
	VANTAGGI DEI LINGUAGGI AD ALTO LIVELLO	1-8
	SVANTAGGI DEI LINGUAGGI AD ALTO LIVELLO	1-8
	LINGUAGGI AD ALTO LIVELLO PER MICROELABORATORI	1-9
	CHE LIVELLO DOVRETE USARE?	1-11
	E PER IL FUTURO?	1-12
	PERCHÈ QUESTO LIBRO?	1-12
	BIBLIOGRAFIA	1-12
2	ASSEMBLATORI	2-1
	CARATTERISTICHE DEGLI ASSEMBLATORI	2-1
	ISTRUZIONI DELL'ASSEMBLATORE	2-1
	LABEL	2-2
	CODICI OPERATIVI DELL'ASSEMBLATORE (MNEMONICA)	2-4
	PSEUDO-OPERAZIONI	2-4
	LA PSEUDO-OPERAZIONE DATA	2-5
	LA PSEUDO-OPERAZIONE EQUATE (O DEFINE)	2-6
	LA PSEUDO-OPERAZIONE ORIGIN	2-7
	LA PSEUDO-OPERAZIONE RESERVE	2-8
	PSEUDO-OPERAZIONI DI LINKING	2-8
	PSEUDO-OPERAZIONI DI GOVERNO INTERNO	2-9
	LABEL CON PSEUDO-OPERAZIONI	2-9
	INDIRIZZI E CAMPO OPERANDO	2-9
	ASSEMBLAGGIO CONDIZIONATO	2-11
	MACRO	2-12
	COMMENTI	2-13
	TIPI DI ASSEMBLATORI	2-14
	ERRORI	2-15
	CARICATORI	2-15
	BIBLIOGRAFIA	2-16
3	L'INSIEME DI ISTRUZIONI DEL LINGUAGGIO ASSEMBLY DELLO Z80	3-1
	REGISTRI DELLA CPU E FLAG DI STATO	3-2
	MODI DI INDIRIZZAMENTO DELLA MEMORIA CON LO Z80	3-4
	IMPLICITO	3-5
	TRASFERIMENTO DI BLOCCO IMPLICITO CON AUTO-INCREMENTO/DECREMENTO	3-7
	STACK IMPLICITO	3-8
	INDICIZZATO	3-10
	DIRETTO	3-11
	RELATIVO AL PROGRAMMA	3-12
	PAGINA BASE	3-13
	INDIRETTO CON REGISTRO	3-14
	IMMEDIATO	3-15
	ABBREVIAZIONI	3-18
	MNEMONICA DELLE ISTRUZIONI	3-21

INDICE (Segue)

CAPITOLO		PAGINA
	CODICI OGGETTO DELLE ISTRUZIONI	3-21
	TEMPI DI ESECUZIONE DELLE ISTRUZIONI	3-21
	STATO	3-22
	COMPATIBILITÀ 8080A/Z80	3-167
	CONVENZIONI DELL'ASSEMBLATORE Z80 DELLA ZILOG	3-173
	STRUTTURA DI CAMPO DELL'ASSEMBLATORE	3-173
	LABEL	3-173
	NOMI RISERVATI	3-173
	PSEUDO-OPERAZIONI	3-173
	ESEMPI	3-174
	LABEL CON PSEUDO-OPERAZIONI	3-175
	INDIRIZZI	3-175
	ASSEMBLY CONDIZIONATO	3-177
	MACRO	3-177
4	PROGRAMMI SEMPLICI	4-1
	FORMATO GENERALE DEGLI ESEMPI	4-1
	DIRETTIVE PER I PROBLEMI	4-2
	ESEMPI DI PROGRAMMA	4-3
	COMPLEMENTO AD UNO	4-3
	SOMMA AD 8 BIT	4-4
	SPOSTAMENTO DI UN BIT VERSO SINISTRA	4-6
	MASCHERATURA DEI QUATTRO BIT PIÙ SIGNIFICATIVI	4-6
	AZZERAMENTO DI UNA LOCAZIONE DI MEMORIA	4-7
	FRAZIONAMENTO DELLA PAROLA	4-8
	RICERCA DEL MAGGIORE DI DUE NUMERI	4-9
	ADDIZIONE A 16 BIT	4-12
	TABELLA DEI QUADRATI	4-14
	COMPLEMENTO AD UNO DI NUMERI A 16 BIT	4-16
	PROBLEMI	4-17
	COMPLEMENTO A DUE	4-17
	SOTTRAZIONE AD 8 BIT	4-17
	SPOSTAMENTO VERSO SINISTRA DI DUE BIT	4-17
	MASCHERATURA DEI QUATTRO BIT MENO SIGNIFICATIVI	4-18
	POSIZIONAMENTO AD UNO DI TUTTI I BIT DI UNA LOCAZIONE DI MEMORIA	4-18
	COMPOSIZIONE DI UNA PAROLA	4-18
	RICERCA DEL MINORE DI DUE NUMERI	4-18
	ADDIZIONE A 24 BIT	4-18
	SOMMA DI QUADRATI	4-19
	COMPLEMENTO A DUE DI UNA PAROLA A 16 BIT	4-19
5	SEMPLICI CICLI DI PROGRAMMA	5-1
	ESEMPI	5-3
	SOMMA DI DATI	5-3
	SOMMA DI DATI A 16 BIT	5-6
	NUMERO DI ELEMENTI NEGATIVI	5-8
	RICERCA DEL MASSIMO	5-11
	GIUSTIFICAZIONE DI UNA FRAZIONE BINARIA	5-14
	PROBLEMI	5-17
	SOMMA DI CONTROLLO DI DATI (CHEKSUM)	5-17
	SOMMA DI DATI A 16 BIT	5-17
	NUMERO DI ELEMENTI UGUALI A ZERO, POSITIVI E NEGATIVI	5-18
	RICERCA DEL MINIMO	5-18
	CONTEGGIO DEI BIT UGUALE AD UNO	5-18

INDICE (Segue)

CAPITOLO		PAGINA
6	DATI CODIFICATI COME CARATTERI	6-1
	ESEMPI	6-2
	LUNGHEZZA DI UNA STRINGA DI CARATTERI	6-2
	RICERCA DEL PRIMO CARATTERE DIVERSO DA BLANK	6-8
	SOSTITUZIONE DEGLI ZERI DI TESTA CON CARATTERI DI BLANK	6-11
	AGGIUNTA DELLA PARITÀ DI TIPO PARI A CARATTERI ASCII	6-13
	CONFRONTO CON STRINGA CAMPIONE	6-16
	PROBLEMI	6-19
	LUNGHEZZA DI UN MESSAGGIO PER TELESCRIVENTE	6-19
	RICERCA DELL'ULTIMO CARATTERE DIVERSO DA BLANK	6-19
	RIDUZIONE DI STRINGA DECIMALE ALLA SOLA PARTE INTERA	6-20
	CONTROLLO DI PARITÀ DI TIPO PARI NEI CARATTERI ASCII	6-21
	CONFRONTO FRA STRINGHE	6-21
7	CONVERSIONE DI CODICE	7-1
	ESEMPI	7-1
	DA ESADECIMALE AD ASCII	7-1
	DA DECIMALE A SETTE SEGMENTI	7-3
	DA ASCII A DECIMALE	7-9
	DA BCD A BINARIO	7-11
	CONVERSIONE DI UN NUMERO BINARIO IN UNA STRINGA ASCII	7-13
	PROBLEMI	7-15
	DA ASCII AD ESADECIMALE	7-15
	DA SETTE-SEGMENTI A DECIMALE	7-15
	DA DECIMALE AD ASCII	7-15
	DA BINARIO A BCD	7-16
	DA STRINGA ASCII A NUMERO BINARIO	7-16
	BIBLIOGRAFIA	7-16
8	PROBLEMI ARITMETICI	8-1
	ESEMPI	8-1
	ADDIZIONE DI MULTI-PRECISIONE	8-1
	TRASFERIMENTO DI BLOCCO	8-4
	ADDIZIONE DECIMALE	8-5
	MOLTIPLICAZIONE BINARIA AD 8 BIT	8-8
	DIVISIONE BINARIA AD 8 BIT	8-13
	NUMERI DI AUTOCONTROLLO DOUBLE ADD	8-18
	DOUBLE MOD 10	8-18
	PROBLEMI	8-26
	SOTTRAZIONE IN MULTI-PRECISIONE	8-26
	SOTTRAZIONE DECIMALE	8-27
	MOLTIPLICAZIONE BINARIA DI 16 BIT PER 8 BIT	8-27
	DIVISIONE BINARIA CON SEGNO	8-28
	NUMERI DI AUTOCONTROLLO «ALIGNED, 1, 3, 7 MOD 10»	8-28
	BIBLIOGRAFIA	8-30
9	TABELLE E LISTE	9-1
	ESEMPI	9-1
	AGGIUNTA DI UN INGRESSO AD UNA LISTA	9-1
	CONTROLLO DI UNA LISTA ORDINATA	9-5
	RIMOZIONE DI UN ELEMENTO DA UNA FILA	9-8
	CLASSIFICAZIONE DI PAROLE AD 8 BIT	9-10
	UTILIZZAZIONE DI UNA TABELLA DI SALTO ORDINATA	9-14
	PROBLEMI	9-16
	RIMOZIONE DI UN INGRESSO DA UNA LISTA	9-16
	AGGIUNTA DI UN INGRESSO AD UNA LISTA ORDINATA	9-17

INDICE (Segue)

CAPITOLO		PAGINA
	AGGIUNTA DI UN ELEMENTO AD UNA FILA	9-17
	CLASSIFICAZIONE DI PAROLE A 16 BIT	9-18
	UTILIZZAZIONE DI UNA TABELLA DI SALTO CON UNA CHIAVE	9-18
	BIBLIOGRAFIA	9-19
10	SUBROUTINE	10-1
	DOCUMENTAZIONE DELLA SUBROUTINE	10-2
	ESEMPI	10-2
	CONVERSIONE DA ESADECIMALE AD ASCII	10-3
	LUNGHEZZA DI UNA STRINGA DI CARATTERI	10-6
	AGGIUNTA DELLA PARITÀ DI TIPO PARI A CARATTERI ASCII	10-10
	CONFRONTO CON CAMPIONE	10-13
	ADDIZIONE IN MULTI-PRECISIONE	10-17
	PROBLEMI	10-19
	DA ASCII AD ESADECIMALE	10-20
	LUNGHEZZA DI UN MESSAGGIO ASCII	10-20
	CONTROLLO DELLA PARITÀ DI TIPO PARI IN CARATTERI ASCII	10-20
	CONFRONTO TRA STRINGHE	10-21
	SOTTRAZIONE DECIMALE	10-22
	BIBLIOGRAFIA	10-22
11	INGRESSO/USCITA	11-1
	INTERVALLI DI TEMPO (RITARDI)	11-8
	ROUTINE DI RITARDO	11-8
	ESEMPIO	11-9
	PROGRAMMA DI RITARDO UTILIZZANTE GLI ACCUMULATORI	11-9
	SEMPLICI DISPOSITIVI DI I/O	11-11
	IL DISPOSITIVO PARALLELO DI INGRESSO/USCITA DELLO Z80 (PIO)	11-11
	CONTROLLO DEL MODO DEL PIO	11-15
	CONFIGURAZIONE DEL PIO	11-17
	ISTRUZIONI D'INGRESSO/USCITA DELLO Z80	11-18
	ESEMPI	11-22
	UN INTERRUTTORE A PULSANTE	11-22
	UN INTERRUTTORE DI TIPO «TOGGLE»	11-28
	INTERRUTTORE A POSIZIONE MULTIPLA (A ROTAZIONE, A SELETTORE O A THUMBWHEEL)	11-33
	UN SINGOLO LED	11-40
	DISPLAY LED A SETTE SEGMENTI	11-43
	PROBLEMI	11-55
	UN PULSANTE APERTO-CHIUSO	11-55
	ANNULLAMENTO DEL RIMBALZO DI UN INTERRUTTORE IN SOFTWARE	11-55
	CONTROLLO DI UN INTERRUTTORE A ROTAZIONE	11-55
	LUCI DI SEGNALAZIONE PER LE POSIZIONI DEGLI INTERRUTTORI	11-56
	CONTARE SU UN DISPLAY A SETTE SEGMENTI	11-56
	DISPOSITIVI DI I/O PIÙ COMPLESSI	11-57
	ESEMPI	11-60
	UNA TASTIERA NON CODIFICATA	11-60
	UNA TASTIERA CODIFICATA	11-69
	UN CONVERTITORE DIGITALE-ANALOGICO	11-72
	UN CONVERTITORE ANALOGICO-DIGITALE	11-76
	TELESCRIVENTE (TTY)	11-81
	IL DISPOSITIVO DI I/O SERIALE DELLO Z80 (SIO)	11-89

INDICE (Segue)

CAPITOLO		PAGINA
	ESEMPI	11-98
	I/O DI UNA TELESCRIVENTE MEDIANTE UN USART	11-98
	INTERFACCE STANDARD	11-103
	PROBLEMI	11-103
	SEPARAZIONE DELLE CHIUSURE DA UNA TASTIERA NON CODIFICATA	11-103
	LEGGERE UNA FRASE DA UNA TASTIERA CODIFICATA	11-103
	GENERATORE D'ONDA QUADRA DI AMPIEZZA VARIABILE	11-104
	MEDIA DI LETTURE ANALOGICHE	11-104
	UN TERMINALE A 30 CARATTERI AL SECONDO	11-104
	BIBLIOGRAFIA	11-105
12	INTERRUPT	12-1
	SISTEMA DI INTERRUPT DELLO Z80	12-3
	INTERRUPT NON MASCHERABILE	12-4
	MODI D'INTERRUPT DELLO Z80	12-4
	COMPATIBILITÀ DELL'INTERRUPT Z80/8080	12-6
	INTERRUPT DEL PIO	12-6
	ESEMPI	12-8
	INTERRUPT DEL SIO	12-10
	ESEMPI DI INTERRUPT	12-13
	UN INTERRUPT DI AVVIAMENTO	12-13
	UN INTERRUPT DA TASTIERA	12-16
	UN INTERRUPT DA STAMPANTE	12-19
	UN INTERRUPT DA UN CLOCK IN TEMPO REALE	12-22
	UN INTERRUPT DA TELESCRIVENTE	12-28
	ROUTINE DI SERVIZIO PIÙ GENERALI	12-33
	PROBLEMI	12-34
	UN INTERRUPT DI TEST	12-34
	UN INTERRUPT DA TASTIERA	12-34
	UN INTERRUPT DA STAMPANTE	12-35
	UN INTERRUPT DA CLOCK IN TEMPO REALE	12-35
	UN INTERRUPT DA TELESCRIVENTE	12-35
	BIBLIOGRAFIA	12-36
13	DEFINIZIONE DEL PROBLEMA E PROGETTO DEL PROGRAMMA	13-1
	I COMPITI DELLO SVILUPPO DEL SOFTWARE	13-1
	DEFINIZIONE DEGLI STADI	13-3
	DEFINIZIONE DEL PROBLEMA	13-4
	DEFINIZIONE DEGLI INGRESSI	13-4
	DEFINIZIONE DELLE USCITE	13-4
	SEZIONE DI ELABORAZIONE	13-5
	MANIPOLAZIONE DEGLI ERRORI	13-5
	FATTORI UMANI	13-6
	ESEMPI	13-6
	RISPOSTA AD UN INTERRUTTORE	13-6
	UN CARICATORE DI MEMORIA BASATO SU INTERRUTTORI	13-8
	UN TERMINALE DI VERIFICA	13-11
	ANALISI DELLA DEFINIZIONE DEL PROBLEMA	13-15
	PROGETTAZIONE DEL PROGRAMMA	13-16
	DIAGRAMMI DI FLUSSO	13-17
	ESEMPI	13-19
	RISPOSTA AD UN INTERRUTTORE	13-19
	IL CARICATORE DI MEMORIA BASATO SU INTERRUTTORI	13-20
	IL TERMINALE DI VERIFICA DI CREDITO	13-22

INDICE (Segue)

CAPITOLO		PAGINA
	PROGRAMMAZIONE MODULARE	13-26
	ESEMPI	13-28
	RISPOSTA AD UN INTERRUTTORE	13-28
	IL CARICATORE DI MEMORIA BASATO SU INTERRUTTORI	13-28
	IL TERMINALE DI VERIFICA	13-28
	ANALISI DELLA PROGRAMMAZIONE MODULARE	13-30
	PROGRAMMAZIONE STRUTTURATA	13-30
	ESEMPI	13-36
	RISPOSTA AD UN INTERRUTTORE	13-36
	IL CARICATORE DI MEMORIA BASATO SU INTERRUTTORI	13-36
	IL TERMINALE DI VERIFICA DI CREDITO	13-38
	ANALISI DELLA PROGRAMMAZIONE STRUTTURATA	13-43
	PROGETTAZIONE TOP-DOWN	13-44
	ESEMPI	13-45
	RISPOSTA AD UN INTERRUTTORE	13-45
	IL CARICATORE DI MEMORIA BASATO SU INTERRUTTORI	13-46
	IL TERMINALE DI TRANSAZIONE	13-47
	ANALISI DELLA PROGETTAZIONE TOP-DOWN	13-49
	ANALISI DELLA DEFINIZIONE DEL PROBLEMA E DEL PROGETTO DEL PROGRAMMA	13-49
	BIBLIOGRAFIA	13-50
14	DEBUGGING E COLLAUDO	14-1
	STRUMENTI SEMPLICI PER IL DEBUGGING	14-1
	STRUMENTI PIÙ AVANZATI PER IL DEBUGGING	14-8
	DEBUGGING CON LISTE DI CONTROLLO	14-10
	RICERCA DEGLI ERRORI	14-11
	ESEMPIO 1 DI DEBUGGING: CONVERSIONE DA DECIMALE A SETTE SEGMENTI	14-16
	ESEMPIO 2 DI DEBUGGING: CLASSIFICAZIONE IN ORDINE DECRESCENTE	14-21
	INTRODUZIONE AL COLLAUDO	14-27
	SCELTA DEI DATI DI COLLAUDO	14-28
	ESEMPIO 1 DI COLLAUDO: PROGRAMMA DI CLASSIFICAZIONE	14-29
	ESEMPIO 2 DI COLLAUDO: NUMERI DI AUTOCONTROLLO	14-29
	PRECAUZIONI NEL COLLAUDO	14-30
	CONCLUSIONI	14-31
	BIBLIOGRAFIA	14-32
15	DOCUMENTAZIONE E RIPROGETTAZIONE	15-1
	PROGRAMMI CHE SI AUTO-DOCUMENTANO	15-1
	COMMENTI	15-2
	ESEMPIO 1 DI COME FARE COMMENTI: ADDIZIONE IN PRECISIONE MULTIPLA	15-4
	ESEMPIO 2 DI COME FARE COMMENTI: USCITA DI TELESCRIVENTE	15-6
	DIAGRAMMI DI FLUSSO COME DOCUMENTAZIONE	15-8
	LINGUAGGI A PROGRAMMAZIONE STRUTTURATA COME DOCUMENTAZIONE	15-8
	MAPPE DI MEMORIA	15-8
	LISTE DI PARAMETRI E DI DEFINIZIONE	15-9
	ROUTINE DI LIBRERIA	15-11
	ESEMPI DI LIBRERIA	15-11
	ESEMPIO 1 DI LIBRERIA: SOMMA DI DATI	15-11
	ESEMPIO 2 DI LIBRERIA: CONVERSIONE DECIMALE/SETTE-SEGMENTI	15-12

INDICE (Segue)

CAPITOLO		PAGINA
	ESEMPIO 3 DI LIBRERIA: SOMMA DECIMALE	15-13
	DOCUMENTAZIONE TOTALE	15-14
	RIPROGETTAZIONE	15-16
	RIORGANIZZAZIONE PER USARE MENO MEMORIA	15-17
	MAGGIORE RIORGANIZZAZIONE	15-19
	BIBLIOGRAFIA	15-20
16	PROGETTI CAMPIONE	16-1
	PROGETTO # 1: UN CRONOMETRO DIGITALE	16-1
	PROGETTO # 2: UN TERMOMETRO DIGITALE	16-15
	BIBLIOGRAFIA	16-29

ELENCO DELLE FIGURE

FIGURA		PAGINA
5-1	Diagramma di flusso di un Loop di Programma	5-2
5-2	Un loop di Programma che permette zero Iterazioni	5-3
7-1	Disposizione a Sette-Segmenti	7-3
8-1	Esempi di Scorrimenti di Digit dello Z80	8-24
11-1	Un Demultiplexer d'Uscita Controllato da un Contatore	11-3
11-2	Un Demultiplexer d'Uscita Controllato da una Porta	11-3
11-3	Un Multiplexer d'Ingresso Controllato da un Contatore	11-4
11-4	Un Multiplexer d'Ingresso Controllato da una Porta	11-4
11-5	Un Handshake d'Ingresso	11-6
11-6	Un Handshake d'Uscita	11-7
11-7	Schema a Blocchi del PIO	11-12
11-8	Schema a Blocchi delle Porte del PIO	11-13
11-9	Controllo di Modo per il PIO Z80	11-16
11-10	Un Circuito con Pulsante	11-22
11-11	Un Circuito con Interruttore con Contatto di Scambio	11-29
11-12	Un Circuito Antirimbalzi Basato su Porte NAND accoppiate tra di loro	11-29
11-13	Un Interruttore a Posizioni Multiple	11-33
11-14	Un Interruttore a Posizioni Multiple con Codificatore	11-34
11-15	Interfacciamento con LED	11-41
11-16	Interfacciamento con Visualizzatore Sette-Segmenti	11-43
11-17	Organizzazione del Visualizzatore a Sette-Segmenti	11-44
11-18	Rappresentazione su Sette-Segmenti di Cifre Decimali	11-46
11-19	Visualizzatori a Sette-Segmenti collegati in Multiplex	11-52
11-20	Una Piccola Tastiera	11-61
11-21	Una Matrice per Tastiera	11-61
11-22	Disposizione di I/O per Scansione di Tastiera	11-62
11-23	Interfacce di I/O per Tastiera Codificata	11-69
11-24	Convertitore D/A NE5018 della Signetics	11-73
11-25	Interfaccia per un Convertitore Digitale-Analogico ad 8 bit	11-74
11-26	Convertitore A/D 8703 della Teledyne	11-77
11-27	Interfaccia per un Convertitore Analogico-Digitale a 8 bit	11-78
11-28	Formato dei Dati di Telescrivente	11-81
11-29	Diagramma di Flusso per Procedure di Ricezione	11-82
11-30	Diagramma di Flusso per Procedure di Trasmissione	11-86
11-31	Schema a Blocchi del SIO Z80	11-90
11-32	Schema a Blocchi di un Canale del SIO	11-91
11-33	Controllo del SIO o Registri di Scrittura	11-92
11-34	Stato del SIO o Registri di Lettura	11-95
12-1	Costituzione di un Vettore d'Interrupt nel Modo 2 d'Interrupt	12-6
12-2	Formato d'una Parola di Controllo d'Interrupt del PIO	12-7
12-3	Formato di una Maschera d'Interrupt del PIO	12-7
13-1	Diagramma di Flusso dello Sviluppo del Software	13-2
13-2	Il Sistema di Interruttori e di Indicatori luminosi	13-7
13-3	Il Caricatore di Memoria Basato su Interruttori	13-9
13-4	Schema a Blocchi di Un Terminale di Controllo	13-11
13-5	Tastiera di Terminale di Controllo	13-12
13-6	Visualizzazione per Terminale di Controllo	13-12
13-7	Simboli Standard di un Diagramma di Flusso	13-17
13-8	Diagramma di Flusso di Risposta per un secondo ad un Interruttore	13-20
13-9	Diagramma di Flusso di un Caricatore di Memoria Basato su di un Interruttore	13-21
13-10	Diagramma di Flusso di un Processo di Entry con Tastiera	13-22
13-11	Diagramma di Flusso di un Processo di Entry con Tastiera con Tasto di Trasmissione	13-23

ELENCO DELLE FIGURE (continua)

FIGURA		PAGINA
13-12	Diagramma di Flusso di un Processo di Entry con Tastiera con Tasti di Funzione	13-24
13-13	Diagramma di Flusso di una Routine di Ricezione	13-25
13-14	Diagramma di Flusso di un Programma Non Strutturato	13-31
13-15	Diagramma di Flusso della Struttura IF — Then — Else	13-32
13-16	Diagramma di Flusso della Struttura Do-While	13-32
13-17	Diagramma di Flusso della Struttura Do-Until	13-33
13-18	Diagramma di Flusso della Struttura Case	13-34
13-19	Diagramma di Flusso Iniziale Per Un Terminale di Transazione	13-47
13-20	Diagramma di Flusso per una Routine KEYBOARD Ampliata	13-48
14-1	Una Semplice Routine con BreakPoint (punto di arresto)	14-2
14-2	Diagramma di Flusso di un Programma di Dump di Registri	14-4
14-3	Risultati di un Tipico Dump di Registri dello Z80	14-6
14-4	Risultati di un Tipico Dump di Memoria	14-8
14-5	Diagramma di Flusso di una Conversione da Decimale a Sette-Segmenti	14-15
14-6	Diagramma di Flusso di un Programma di Ricerca	14-20
16-1	Configurazione di I/O per un Cronometro Digitale	16-2
16-2	Configurazione di I/O per un Termometro Digitale	16-16
16-3	Hardware Analogico di un Termometro Digitale	16-17
16-4	Caratteristiche di un Termistore (Fenwal GA51J1 Bead)	16-18
16-5	Tipica curva E-I per il Termistore (25°C)	16-19

ELENCO DELLE TABELLE

TABELLA		PAGINA
1-1	Tabella di Conversione Esadecimale	1-4
2-1	I Campi di un'Istruzione in Linguaggio Assembly	2-1
2-2	Delimitatori Standard dell'Assemblatore dello Z80	2-2
2-3	Assegnazione ed Utilizzazione di una Label	2-3
3-1	Istruzioni dello Z80 Usate Frequentemente	3-16
3-2	Istruzioni dello Z80 Usate Occasionalmente	3-17
3-3	Istruzioni dello Z80 Usate Raramente	3-18
3-4	Sommario dell'Insieme di Istruzioni dello Z80	3-23
3-5	Codici Oggetto delle Istruzioni in Ordine Numerico	3-40
3-6	Corrispondenze tra i Registri ed i Flag dello Z80 e dell'8080A	3-44
3-7	Corrispondenza tra i Codici Mnemonici dell'8080A e dello Z80	3-45
3-8	Corrispondenza tra i Codici Mnemonici dello Z80 e dell'8080A	3-46
3-9	Codici Operativi Inutilizzati dell'8080A e Loro Significato per lo Z80	3-48
6-1	Tabella Esadecimale-ASCII	6-2
11-1	Indirizzi del PIO	11-14
11-2	Indirizzamento dei Registri di Controllo del PIO	11-14
11-3	Ingresso dei Dati rispetto alla Posizione dell'Interruttore	11-34
11-4	Rappresentazione su Sette-Segmenti di Numeri Decimali	11-45
11-5	Rappresentazioni su Sette-Segmenti di Lettere e Simboli	11-48
11-6	Confronto tra Collegamenti Singoli e Collegamenti a Matrice per Tastiera	11-60
11-7	Indirizzi del SIO	11-90
12-1	L'istruzione Restart (RST)	12-4
14-1	Restart dello Z80 ed Indirizzi degli Interrupt	14-2

INDICE PER ARGOMENTI

INDICE		PAGINA
A	ABILITAZIONE E DISABILITAZIONE DEGLI INTERRUPT DEL PIO	12-8
	ACCESSO DIRETTO IN MEMORIA	11-5
	ADATTAMENTO DECIMALE	8-7
	ALGORITMI MIGLIORI	15-19
	ALGORITMO DELLA DIVISIONE	8-13
	ALGORITMO DI MOLTIPLICAZIONE	8-8
	ALLOCAZIONE DI RAM	2-8
	ALTRE VARIAZIONI DI MAGGIORE IMPORTANZA	15-19
	ALTRI SISTEMI NUMERICI	2-10
	ANALIZZATORE LOGICO	14-9
	APPLICAZIONI PER LINGUAGGIO AD ALTO LIVELLO	1-11
	APPLICAZIONI PER LINGUAGGIO ASSEMBLY	1-11
	APPLICAZIONI PER LINGUAGGIO DI MACCHINA	1-11
	AREE DI APPLICAZIONE PER LIVELLI DI LINGUAGGIO	1-10
	ARROTONDAMENTO BINARIO	8-25
	ARROTONDAMENTO DECIMALE	8-25
	ASSEMBLAGGIO MANUALE	1-5
	ASSEMBLATORE	1-6
	ASSEMBLATORE A DUE PASSAGGI	2-15
	ASSEMBLATORE AD UN PASSAGGIO	2-14
	ASSEMBLATORE RESIDENTE	2-14
	ATTESA DELLA CHIUSURA DI UN TASTO	11-63
	AUSILI PER IL COLLAUDO	14-27
B	BUFFERING DOPPIO	12-19
C	CAMBIAMENTO DELL'INDIRIZZO DI RITORNO	12-17
	CAMBIAMENTI DI STATO CON L'ESECUZIONE DI ISTRUZIONI	3-22
	CAMPI DEL LINGUAGGIO ASSEMBLY	2-1
	CAMPO DI LABEL	2-2
	CARATTERI ASCII	2-11
	CARATTERISTICHE DEI MODI DEL PIO	11-16
	CARATTERISTICHE DEI SISTEMI DI INTERRUPT	12-1
	CARATTERISTICHE DI COMPATIBILITÀ 8080A/Z80	3-167
	CARATTERISTICHE IMPORTANTI DEGLI ANALIZZATORI LOGICI	14-10
	CARATTERISTICHE SPECIALI DEL SIO	11-97
	CARICATORE BOOTSTRAP	2-15
	CARICATORE DI RIALLOCAZIONE	2-15
	CARICATORE ESADECIMALE	1-4
	CARICATORI DI LINKING	2-15
	CATEGORIE DI I/O	11-1
	CHE COSA INCLUDERE NEL LISTING DI CONTROLLO	14-10
	CLOCK IN TEMPO REALE	12-22
	CODICI OPERATIVI A 2 PAROLE	3-167
	CODICI OPERATIVI NON UTILIZZATI DELL'8080A	3-167
	CODIFICA	13-3
	COLLAUDO	13-3
	COLLAUDO DI UN PROGRAMMA ARITMETICO	14-29
	COLLAUDO DI UN PROGRAMMA DI CLASSIFICAZIONE	14-29
	COLLAUDO DI PROCESSI SPECIALI	14-28
	COLLAUDO STRUTTURATO	14-28
	COMBINAZIONE DELLE INFORMAZIONI DI CONTROLLO	11-58
	COMPILATORE	1-7
	COMPITI PER ROUTINE DI SERVIZIO GENERALI	12-33
	CONSERVAZIONE DEL TEMPO REALE	12-26
	CONSIDERAZIONI DI ERRORE	13-5

INDICE PER ARGOMENTI (Segue)

INDICE		PAGINA
	CONTATORE DI LOCAZIONE	2-7
	CONTROLLO DEL LED	11-40
	CONVERSIONE DI LIVELLO DI ASSEMBLY 8080A/Z80	3-167
	CORREZIONE DEGLI ERRORI DI TASTIERA	13-14
	CORREZIONE DEGLI ERRORI DI TRASMISSIONE	13-15
	CORREZIONE DELL'ERRORE DI OPERATORE IN UN CARICATORE DI MEMORIA	13-10
	COSTANTE DEL LOOP DI RITARDO CON Z80	11-10
	COSTANTE DI RIALLOCAZIONE	2-3
	COSTO DEI COMPILATORI	1-9
	COSTO DI RIPROGETTAZIONE	15-16
	CROSS-ASSEMBLATORE	2-14
D	DATI O INDIRIZZI DECIMALI	2-9
	DEBUGGING	13-3
	DEBUGGING DI PROGRAMMI GESTITI AD INTERRUPT	14-14
	DEBUGGING DI UN PROGRAMMA DI CLASSIFICAZIONE	14-21
	DEBUGGING DI UN PROGRAMMA DI CONVERSIONE DI CODICE	14-16
	DECISIONE SU UNA VARIAZIONE DI GRANDE RILEVANZA	15-19
	DEFINIZIONE DEL PROBLEMA	13-3
	DEFINIZIONE DEL SISTEMA INTERRUTTORE E LAMPADA	13-6
	DEFINIZIONE DI UNA SEQUENZA DI ISTRUZIONI	2-12
	DEFINIZIONE DI UN CARICATORE DI MEMORIA BASATO SU INTERRUTTORI	13-8
	DEFINIZIONE DI UN TERMINALE DI VERIFICA	13-11
	DEFINIZIONI DI NOMI	2-6
	DELIMITATORI	2-2
	DIAGRAMMA DI FLUSSO DELLA VERIFICA DI CREDITO	13-22
	DIAGRAMMA DI FLUSSO DEL CARICATORE DI MEMORIA BASATO SU INTERRUTTORI	13-20
	DIAGRAMMI DI FLUSSO DI DATI	13-19
	DIAGRAMMA DI FLUSSO DI UN SISTEMA CON INTERRUTTORE E LAMPADA	13-19
	DIRETTIVA DI ASSEMBLATORE	2-4
	DIRETTIVE DI PROGRAMMAZIONE	4-2
	DIRETTIVE PER GLI ESEMPI	4-1
	DIREZIONI DEL PIO NEL MODO DI CONTROLLO	11-15
	DISABILITAZIONE DEGLI INTERRUPT	12-28
	DISPLAY AD ANODO COMUNE O A CATODO COMUNE	11-43
	DOCUMENTAZIONE	13-3
	DOCUMENTAZIONE DEI TRASFERIMENTI DI STATO E DI CONTROLLO	11-59
	DOCUMENTAZIONE DELLE SUBROUTINE	10-2
	DOMANDE PER CONTROLLO MANUALE	14-11
	DOMANDE PER I COMMENTI	15-5
	DRIVER DI I/O	11-18
E	ELIMINAZIONE DEI RIMBALZI CON PORTE NAND MUTUAMENTE ACCOPPIATE	11-28
	ELIMINAZIONE DEI RIMBALZI IN SOFTWARE	11-25
	ELIMINAZIONE DI UNO ZERO DI GUIDA	16-22
	ERRORI COMUNI	14-11
	ESECUZIONE DELLE ISTRUZIONI PIÙ VELOCEMENTE E PIÙ LENTAMENTE	3-167
	ESEMPI DI CONFIGURAZIONE DELLE ISTRUZIONI DEL PIO	12-12
	ESEMPIO DI CONFIGURAZIONE DEL SIO	11-100
	ESEMPI DI ISTRUZIONI DI I/O	11-19

INDICE PER ARGOMENTI (Segue)

INDICE		PAGINA
	ESEMPI DI STRUTTURE	13-33
	ESPANSIONE DELLA ROUTINE KEYBOARD	13-48
	ESPRESSIONI ARITMETICHE E LOGICHE	2-11
F	FATTORI IN INGRESSO	13-4
	FATTORI NELLA ELABORAZIONE	13-5
	FLAG DI ADD/SUBTRACT	8-7
	FORMATO	2-2
	FORMATO DEGLI ESEMPI	4-1
	FORMATO DEL CARATTERE	11-81
	FORMATO PER LA PROGETTAZIONE TOP-DOWN	13-49
	FORMAZIONE DI CLASSI DI DATI	14-28
	FORME DI LIBRERIE DI PROGRAMMI STANDARD	15-11
	FORTRAN	1-7
	FREQUENZA DEL CLOCK IN TEMPO REALE	12-22
	FULL-DUPLEX	11-89
	FUNZIONAMENTO DI UN DISPOSITIVO IN DAISY-CHAIN	12-10
G	GESTIONE DEGLI ERRORI DEL CARICATORE DI MEMORIA	13-10
	GESTIONE DEI DATI IN ASCII	6-1
	GESTIONE DEGLI INTERRUPT DA MONITOR	12-14
H	HANDSHAKE	11-2
	HARDWARE ANALOGICO DEL TERMOMETRO	16-15
	HASHING	9-4
I	IMPORTANZA RELATIVA DELLA CODIFICA	13-1
	INADATTABILITÀ DEI LINGUAGGI AD ALTO LIVELLO	1-10
	INCOMPATIBILITÀ DI TEMPORIZZAZIONE	3-168
	INCOMPATIBILITÀ 8080A/Z80	3-167
	INCOMPATIBILITÀ 8085/Z80	3-168
	INDICAZIONI PER I COMMENTI	15-2
I (Segue)	INDIPENDENZA DALLA MACCHINA DEI LINGUAGGI AD ALTO LIVELLO	1-8
	INDIRIZZAMENTO DEI REGISTRI DI LETTURA E DI SCRITTURA DEL SIO	11-89
	INDIRIZZI DEL PIO	11-11
	INDIRIZZI DEL SIO	11-89
	INEFFICIENZA DEI LINGUAGGI AD ALTO LIVELLO	1-9
	INFORMAZIONI DI CONTROLLO E DI STATO	11-58
	INGRESSO PER INTERRUTTORE E LAMPADA	13-6
	INGRESSI DEL TERMINALE DI VERIFICA	13-13
	INGRESSI DI INTERRUPT DELLO Z80	12-3
	INIZIALIZZAZIONE DELLA RAM	2-8
	INTERFACCIA CON TTY	11-81
	INTERFACCIAMENTO CON DISPOSITIVI AD ALTA VELOCITÀ	11-5
	INTERFACCIAMENTO CON DISPOSITIVI A MEDIA VELOCITÀ	11-2
	INTERFACCIAMENTO CON DISPOSITIVI LENTI	11-2
	INTERFACCE STANDARD	11-103
	INSERIMENTO DI PUNTI DI ARRESTO	14-3
	INTERAZIONE DELL'OPERATORE	13-6
	INTERRUPT DA TASTIERA	12-16
	INTERRUPT NON MASCHERABILE	12-2
	INTERRUPT NON MASCHERABILE DELLO Z80	12-4
	INTERRUPT DEL PIO	12-6
	INTERRUPT DEL PIO IN DAISY-CHAIN	12-9
	INTERRUPT DEL SIO	12-10
	INTERRUPT SU PARTICOLARI MICROCOMPUTER	12-14
	ISTRUZIONE DI INTERRUPT DELLO Z80	12-3
	ISTRUZIONE RESTART	12-5

INDICE PER ARGOMENTI (Segue)

INDICE		PAGINA
	ISTRUZIONI BINARIE	1-1
	ISTRUZIONI DI ESPLORAZIONE DI BLOCCO	6-6
	ISTRUZIONI DI SCORRIMENTO DECIMALI	8-22
	ISTRUZIONI DI TRASFERIMENTO DI BLOCCO	8-4
	ISTRUZIONI PER LE SUBROUTINE	10-1
	ISTRUZIONI DI I/O A BLOCCHI	11-19
	ISTRUZIONI DI I/O CON INDIRIZZAMENTO ASSOLUTO	11-18
	ISTRUZIONI DI I/O CON INDIRIZZAMENTO INDIRETTO	11-18
	ISTRUZIONI DI I/O DELLO Z80	11-18
	ISTRUZIONI SPECIALI	4-3
	I/O E MEMORIA	11-1
L	LABEL NELLE ISTRUZIONI DI SALTO	2-2
	LIBRERIA DI SUBROUTINE	10-1
	LIMITI DEL MODO PASSO SINGOLO	14-2
	LINEE DI CONTROLLO E REGISTRI DEL PIO	11-11
	LINK EDITOR	2-16
M	MACRO-ASSEMBLATORE	2-14
	MANIPOLAZIONE DEGLI ERRORI DI INTERRUTTORE E LAMPADA	13-7
	MANIPOLAZIONE DEGLI ERRORI DEL TERMINE DI VERIFICA	13-14
	MANUTENZIONE E RIPROGETTAZIONE	13-3
	META-ASSEMBLATORE	2-14
	METODI DI RICERCA	9-6
	METODI DI PROGETTAZIONE TOP-DOWN	13-44
	METODI PER PRODURRE INTERVALLI DI TEMPO	11-8
	MICRO-ASSEMBLATORE	12-12
	MISURA DELLO STATO DI AVANZAMENTO NEGLI STADI	13-1
	MODI DEL PIO	11-15
	MODI DI INTERRUPT	12-4
	MODI DI USCITA DEL PIO	11-15
	MODO BIDIREZIONALE DEL PIO	11-15
	MODO DI CONTROLLO DEL PIO	11-15
	MODO DI CONTROLLO DI INTERRUPT DEL PIO	12-7
	MODO DI INGRESSO DEL PIO	11-15
	MODO DI RICEZIONE DELLA TTY	11-81
	MODO DI TRASMISSIONE DELLA TTY	11-86
	MODULARIZZAZIONE DEL CARICATORE DI MEMORIA BASATO SU INTERRUTTORI	13-28
	MODULARIZZAZIONE DEL SISTEMA CON INTERRUTTORE E LAMPADA	13-28
	MODULAZIONE DEL TERMINALE DI VERIFICA	13-28
N	NOTAZIONE ALGEBRICA	1-9
	NUMERI CHE SI AUTOCONTROLLANO	8-18
	NUMERI E CARATTERI NEL CAMPO INDIRIZZI	3-175
O	OPERAZIONI LOGICHE ED ARITMETICHE DI ASSEMBLATORE	3-175
	ORIENTAMENTI FUTURI NEI LIVELLI DI LINGUAGGIO	1-12
	OTTALE O ESADECIMALE	1-3
P	PASSAGGIO DI PARAMETRI	10-1
	PASSI PER CONFIGURARE UN PIO	11-17
	PASSO SINGOLO	14-1
	POLLING	12-2
	PORTABILITÀ	1-7
	PORTABILITÀ DEI LINGUAGGI AD ALTO LIVELLO	1-8
	POSIZIONAMENTO DELLE DEFINIZIONI	2-7
	PRECISIONE DECIMALE IN BINARIO	8-4
	PRECISIONE IN BINARIO E IN BCD	8-8

INDICE PER ARGOMENTI (Segue)

INDICE		PAGINA
	PRINCIPI BASILARI DELLA PROGETTAZIONE DI UN PROGRAMMA	13-16
	PRINCIPI DI MODULARIZZAZIONE	13-27
	PRINCIPIO DI CELARE LE INFORMAZIONI	13-29
	PRIORITÀ	12-2
	PRIORITÀ DI CLOCK IN TEMPO REALE	12-23
	PROBLEMI DI MNEMONICA	1-5
	PROCEDURA D'INGRESSO DEL CRONOMETRO	16-1
	PROGETTAZIONE BOTTOM-UP	13-44
	PROGETTAZIONE DEL PROGRAMMA	13-3
	PROGETTAZIONE TOP-DOWN DEL SISTEMA DI INTERRUTTORE E LAMPADA	13-45
	PROGETTAZIONE TOP-DOWN DEL TERMINALE DI VERIFICA	13-47
	PROGETTAZIONE TOP-DOWN DI UN CARICATORE DI MEMORIA BASATO SU INTERRUTTORI	13-46
	PROGRAMMA DEL COMPUTER	1-2
	PROGRAMMA IN LINGUAGGIO ASSEMBLY	1-5
	PROGRAMMA IN LINGUAGGIO DI MACCHINA	1-2
	PROGRAMMA OGGETTO	1-2,1-6
	PROGRAMMA SORGENTE	1-6
	PROGRAMMAZIONE STRUTTURATA NEL SISTEMA DI INTERRUTTORE E LAMPADA	13-36
	PROGRAMMAZIONE STRUTTURATA PER IL CARICATORE DI MEMORIA BASATO SU INTERRUTTORI	13-36
	PROGRAMMA STRUTTURATO PER IL TERMINALE DI VERIFICA DI CREDITO	13-38
	PROPAGAZIONE DEL SEGNO	8-26
	PSEUDO-OPERAZIONE DEFS	3-174
	PSEUDO-OPERAZIONE END	3-175
	PSEUDO-OPERAZIONE EQU	3-174
	PSEUDO-OPERAZIONE DEFL	3-174
	PSEUDO-OPERAZIONE ORG	3-174
	PSEUDO-OPERAZIONI	2-4
	PSEUDO-OPERAZIONI COND E ENDC	3-177
	PSEUDO-OPERAZIONI DEFB, DEFM, DEFW	3-173
	PSEUDO-OPERAZIONI MACRO E ENDM	3-177
	PUNTO DI ARRESTO	14-2
Q	QUANDO USARE LINGUAGGI AD ALTO LIVELLO PER PROGRAMMAZIONE STRUTTURATA	13-35
R	RADDOPPIO E DIMEZZAMENTO DI NUMERI BINARI	8-24
	RAGIONAMENTO DIETRO GLI INTERRUPT	12-1
	RAPPRESENTAZIONI A SETTE SEGMENTI	11-45
	REGOLE PER IL COLLAUDO	14-30
	REGOLE PER LA PROGRAMMAZIONE MODULARE	13-30
	REGOLA PER LA PROGRAMMAZIONE STRUTTURATA	13-43
	REGOLE PER LISTE DI DEFINIZIONE	15-9
	REGOLE PER PROGRAMMI CHE SI AUTODOCUMENTANO	15-1
	REGOLE PER SCEGLIERE LABEL	2-3
	RESET DEL SIO	11-97
	RIALLOCAZIONE	10-2
	RIDUZIONE DEGLI ERRORI DI TRASMISSIONE	11-5
	RIEMPIMENTO DI UN BUFFER MEDIANTE INTERRUPT	12-18
	RIFERIMENTI ESTERNI	2-8
	RIMBALZO DI UN INTERRUTTORE	11-25
	RIORGANIZZAZIONE DI MAGGIORE O DI MINORE IMPORTANZA	15-16

INDICE PER ARGOMENTI (Segue)

INDICE		PAGINA
	RISPARMIO DI MEMORIA	15-17
	RISPARMIO DI TEMPO DI ESECUZIONE	15-18
	RISPOSTA AGLI INTERRUPT DELLO Z80	12-3
	RITARDO ELEMENTARE IN SOFTWARE	11-8
	ROLLOVER	11-69
	ROUTINE DI INTERRUPT DEL SIO	12-28
	ROUTINE DI RITARDO TRASPARENTE	11-8
	ROUTINE STRUTTURATA DI RICEZIONE	13-40
	ROUTINE STRUTTURATA DI TASTIERA	13-38
	RST COME PUNTO DI ARRESTO	14-2
S	SALTI INDIRETTI	9-15
	SALVATAGGIO DI VALORI IN REGISTRI CON APICE	12-18
	SCANSIONE DELLA TASTIERA	11-60
	SCARICAMENTO DI MEMORIA	14-7
	SCARICAMENTO DI REGISTRI	14-4
	SCELTA DEI METODI	12-12
	SCELTA DELLE LABEL	2-3
	SCELTA DI NOMI	2-7
	SCELTA DI NOMI UTILI	15-2
	SCELTA DI UN ASSEMBLATORE	1-6
	SCELTA DI UN METODO DI TEMPORIZZAZIONE	11-8
	SCELTE DI DATI DALLE CLASSI	14-29
	SEGNALI DI DAISY-CHAIN DEL PIO	12-10
	SEMPLICE ALGORITMO DI CLASSIFICAZIONE	9-11
	SEPARAZIONE DELLE INFORMAZIONI DI STATO	11-58
	SEZIONI DI DIAGRAMMA DI FLUSSO	13-22
	SIMULATORE IN SOFTWARE	14-8
	SINCRONIZZAZIONE CON CLOCK IN TEMPO REALE	12-22
	SINCRONIZZAZIONE CON DISPOSITIVI DI I/O	11-57
	SINTASSI DEI LINGUAGGI AD ALTO LIVELLO	1-8
	SISTEMI DI INTERRUPT A POLLING CON IL SIO	12-11
	SOMMA AD 8 BIT	5-3
	SPESE GENERALI PER LINGUAGGI AD ALTO LIVELLO	1-10
	SPOSTAMENTI DI DATI IN UN BLOCCO	7-9
	STADI DELLO SVILUPPO DEL SOFTWARE	13-1
	STATO DI ERRORE DEL SIO	11-100
	STROBE	11-5
	STRUTTURA DELLA DOCUMENTAZIONE	15-14
	STRUTTURE DI BASE DELLA PROGRAMMAZIONE STRUTTURATA	13-31
	STUB	13-44
	SUBROUTINE RIENTRANTE	10-2
	SUGGERIMENTI SULL'USO DEI DIAGRAMMI DI FLUSSO	15-8
	SVANTAGGI DEI LINGUAGGI AD ALTO LIVELLO	1-9
	SVANTAGGI DELLA PROGRAMMAZIONE MODULARE	13-27
	SVANTAGGI DELLA PROGRAMMAZIONE STRUTTURATA	13-35
	SVANTAGGI DEGLI INTERRUPT	12-2
	SVANTAGGI DELLE MACRO	2-13
	SVANTAGGI DEI DIAGRAMMI DI FLUSSO	13-18
	SVANTAGGI DI UNA PROGETTAZIONE TOP-DOWN	13-44
	SVILUPPO DEGLI STUB	13-44
	SVUOTAMENTO DI UN BUFFER CON INTERRUPT	12-21
T	TABELLA DEI SIMBOLI	2-6
	TABELLA DEI TASTI	16-7
	TASTIERA A MATRICE	11-60

INDICE PER ARGOMENTI (Segue)

INDICE		PAGINA
	TECNICHE DI COMMENTO	2-13
	TERMINATORI E STRUTTURE	13-43
	TIPICA LISTA DI DEFINIZIONI	15-10
	TTY STANDARD	11-81
U	UART	11-88
	USCITE PER INTERRUTTORE E PER LAMPADA	13-7
	USCITE VERSO IL TERMINALE DI VERIFICA	13-13
	USI DEGLI INTERVALLI DI TEMPO	11-8
	USO DEI REGISTRI INDICE DELLO Z80	7-7
	USO DELLA COPPIA DI REGISTRI HL	4-2
	USO DELLE ISTRUZIONI DI I/O A BLOCCHI	11-21
	USO DELL'ACCUMULATORE	4-2
	USO DI UNA TABELLA DI CALIBRAZIONE	16-21
	USO DI NOMI	2-6
	USO DI UN CODIFICATORE TTL	11-34
	UTILIZZO DI PROCESSI DI COLLAUDO NEL DEBUGGING	14-27
V	VANTAGGI DEI LINGUAGGI AD ALTO LIVELLO	1-9
	VANTAGGI DELLA PROGETTAZIONE TOP-DOWN	13-44
	VANTAGGI DELLA PROGRAMMAZIONE MODULARE	13-26
	VANTAGGI DELLA PROGRAMMAZIONE STRUTTURATA	13-35
	VANTAGGI DELLE MACRO	2-13
	VANTAGGI DEI DIAGRAMMI DI FLUSSO	13-17
	VANTAGGI E SVANTAGGI DEGLI INTERRUPT IN DAISY-CHAIN	12-10
	VARIABILI LOGICHE O GLOBALI	2-13
	VETTORE DI INTERRUPT DEL PIO	12-7
	VETTORIZZAZIONE	12-2

INDICE DESCRIZIONI DELLE ISTRUZIONI

ADC A,data 3-43
ADC A,reg 3-44
ADC A,(HL) 3-45
ADC A,(IX + disp) 3-45
ADC A,(IY + disp) 3-45
ADC HL,rp 3-46
ADD A,data 3-47
ADD A,reg 3-48
ADD A,(HL) 3-49
ADD A,(IX + disp) 3-49
ADD A,(IY + disp) 3-49
ADD HL,rp 3-50
ADD xy,rp 3-51
AND data 3-52
AND reg 3-53
AND (HL) 3-54
AND (IX + disp) 3-54
AND (IY + disp) 3-54

BIT b,reg 3-55
BIT b,(HL) 3-56
BIT b,(IX + disp) 3-56
BIT b,(IY + disp) 3-56

CALL label 3-57
CALL condition,label 3-58
CCF 3-59
CP data 3-60
CP reg 3-61
CP (HL) 3-62
CP (IX + disp) 3-62
CP (IY + disp) 3-62
CPD 3-63
CPDR 3-64
CPI 3-65
CPIR 3-66
CPL 3-67

DAA 3-68
DEC reg 3-69
DEC rp 3-70
DEC IX 3-70
DEC IY 3-70
DEC (HL) 3-71
DEC (IX + disp) 3-71
DEC (IY + disp) 3-71
DI 3-72
DJNZ disp 3-73

EI 3-73
EX AF,AF' 3-75
EX DE,HL 3-76
EX (SP),HL 3-77
EX (SP),IX 3-77
EX (SP),IY 3-77
EXX 3-78

HALT 3-79
IM 0 3-80
IM 1 3-80
IM 2 3-80
IN A,(port) 3-81
INC reg 3-82
INC rp 3-83
INC IX 3-83
INC IY 3-83
INC (HL) 3-84
INC (IX + disp) 3-84
INC (IY + disp) 3-84
IND 3-85
INDR 3-86
INI 3-86
INIR 3-87
IN reg,(C) 3-88

JP label 3-89
JP condition,label 3-90
JP (HL) 3-91
JP (IX) 3-91
JP (IY) 3-91
JR C,disp 3-92
JR disp 3-93
JR NC,disp 3-94
JR NZ,disp 3-94
JR Z,disp 3-95

LD A,I 3-95
LD A,R 3-95
LD A,(addr) 3-96
LD A,(rp) 3-97
LD dst,src 3-98
LD HL,(addr) 3-99
LD rp,(addr) 3-99
LD IX,(addr) 3-99
LD IY,(addr) 3-99
LD I,A 3-100
LD R,A 3-100
LD reg,data 3-101
LD rp,data 3-102
LD IX,data 3-102
LD IY,data 3-102
LD reg,(HL) 3-103
LD reg,(IX + disp) 3-103
LD reg,(IY + disp) 3-103
LD SP,HL 3-104
LD SP,IX 3-104
LD SP,IY 3-104
LD (addr),A 3-105
LD (addr),HL 3-106
LD (addr), rp 3-106
LD (addr),xy 3-106
LD (HL),data 3-108
LD (IX + disp),data 3-108

INDICE DESCRIZIONI DELLE ISTRUZIONI (Segue)

LD (IY + disp),data 3-108
LD (HL),reg 3-109
LD (IX + disp),reg 3-109
LD (IY + disp),reg 3-109
LD (rp),A 3-110
LDD 3-111
LDDR 3-112
LDI 3-113
LDIR 3-114

NEG 3-115
NOP 3-116

OR data 3-117
OR reg 3-118
OR (HL) 3-119
OR (IX + disp) 3-119
OR (IY + disp) 3-119
OUT (C),reg 3-120
OUTD 3-121
OTDR 3-121
OUTI 3-122
OUT (port),A 3-123

POP rp 3-124
POP IX 3-124
POP IY 3-124
PUSH rp 3-125
PUSH IX 3-125
PUSH IY 3-125

RES b,reg 3-126
RES b,(HL) 3-127
RES b,(IX + disp) 3-127
RES b,(IY + disp) 3-127
RET 3-128
RET cond 3-129
RETI 3-130
RETN 3-131
RL reg 3-132
RL (HL) 3-133
RL (IX + disp) 3-133
RL (IY + disp) 3-133
RLA 3-134
RLC reg 3-135
RLC (HL) 3-136
RLC (IX + disp) 3-136
RLC (IY + disp) 3-136
RLCA 3-137
RLD 3-138
RR reg 3-139
RR (HL) 3-140
RR (IX + disp) 3-140
RR (IY + disp) 3-140
RRA 3-141
RRC reg 3-142
RRC (HL) 3-143
RRC (IX + disp) 3-143
RRC (IY + disp) 3-143
RRCA 3-144
RRD 3-145
RST n 3-146

SBC A,data 3-147
SBC A,reg 3-148
SBC A,(HL) 3-149
SBC A,(IX + disp) 3-149
SBC A,(IY + disp) 3-149
SBC HL,rp 3-150
SCF 3-151
SET b,reg 3-152
SET b,(HL) 3-153
SET b,(IX + disp) 3-153
SET b,(IY + disp) 3-153
SLA reg 3-154
SLA (HL) 3-155
SLA (IX + disp) 3-155
SLA (IY + disp) 3-155
SRA reg 3-156
SRA (HL) 3-157
SRA (IX + disp) 3-157
SRA (IY + disp) 3-157
SRL reg 3-158
SRL (HL) 3-159
SRL (IX + disp) 3-159
SRL (IY + disp) 3-159
SUB data 3-161
SUB reg 3-162
SUB (HL) 3-163
SUB (IX + disp) 3-163
SUB (IY + disp) 3-163

XOR data 3-164
XOR reg 3-165
XOR (HL) 3-166
XOR (IX + disp) 3-166
XOR (IY + disp) 3-166

Capitolo 1

INTRODUZIONE ALLA PROGRAMMAZIONE IN LINGUAGGIO ASSEMBLY

Questo libro descrive la programmazione in linguaggio assembly. Si suppone che siate a conoscenza di An Introduction To Microcomputers: Volume 1 — Basic Concepts (in particolare i Capitoli 6 e 7). Questo libro non tratta delle caratteristiche generali di computer, microcomputer, metodi di indirizzamento o insiemi di istruzioni; per quelle informazioni dovreste riferirvi a An Introduction To Microcomputers: Volume 1.

COME È STATO STAMPATO QUESTO LIBRO

Vi facciamo notare che questo libro è stato stampato in neretto e in chiaro. Ciò è stato fatto per aiutarvi per saltare quelle parti del libro che trattano argomenti di cui siete a conoscenza. Potete essere certi che il carattere in chiaro sviluppa solamente le informazioni presentate precedentemente in neretto. Perciò, leggete solo il testo in neretto fino a che non arrivate ad un argomento che volete conoscere di più; a questo punto cominciate a leggere il carattere in chiaro.

IL SIGNIFICATO DELLE ISTRUZIONI

Il set (insieme) delle istruzioni di un microprocessore è l'insieme di ingressi binari che producono azioni ben definite durante un ciclo d'istruzione. Un insieme di istruzioni sta ad un microprocessore come una tabella di funzioni sta ad un dispositivo logico quale una porta, un sommatore o un registro a scorrimento.

Naturalmente le azioni che un microprocessore effettua in risposta agli ingressi delle istruzioni sono molto più complesse delle azioni che dispositivi a logica combinatoria effettuano in risposta ai loro ingressi.

ISTRUZIONI BINARIE

Un'istruzione è semplicemente una combinazione binaria di bit — essa deve essere disponibile agli ingressi dei dati del microprocessore in un istante opportuno per essere interpretata come istruzione. Per esempio, quando il microprocessore Z80 riceve la combinazione binaria ad 8 bit 10000000 come ingresso durante un'operazione di fetch (prelievo) delle istruzioni, la combinazione assume il significato di:

«Somma il contenuto del registro B al contenuto dell'Accumulatore»

Analogamente, la combinazione 00111110 sta a significare:

«Carica nell'Accumulatore il contenuto della prossima parola della memoria dei programmi»

Il microelaboratore (come ogni altro computer) riconosce solo combinazioni binarie come istruzioni o dati; non riconosce parole o numeri ottali, decimali oppure esadecimali.

UN PROGRAMMA DEL COMPUTER

Un programma è una serie di istruzioni che obbliga il computer ad effettuare un particolare compito.

PROGRAMMA DEL COMPUTER

In realtà, un programma del computer comprende qualcosa di più delle istruzioni; contiene pure i dati e gli indirizzi di memoria di cui il computer necessita per compiere i compiti definiti dalle istruzioni. Chiaramente, se il computer deve effettuare un'addizione, deve avere due numeri da sommare ed una destinazione per il risultato. Il programma del computer deve determinare le sorgenti dei dati e della destinazione del risultato così come deve specificare l'operazione da realizzare.

Tutti i microelaboratori eseguono istruzioni in sequenza a meno che una delle istruzioni cambi la sequenza di esecuzione o metta il computer in «halt» (per esempio l'elaboratore prende l'istruzione successiva del successivo indirizzo di memoria consecutivo a meno che l'istruzione corrente non gli dica specificatamente di fare altrimenti).

Infine ogni programma viene tradotto in un insieme di numeri binari. Per esempio, ecco il programma per lo Z80 che somma il contenuto delle locazioni di memoria 60_{16} e 61_{16} e mette il risultato nella locazione 62_{16}:

00111010
01100000
00000000
01000111
00111010
01100001
00000000
10000000
00110010
01100010
00000000

PROGRAMMA OGGETTO

PROGRAMMA IN LINGUAGGIO DI MACCHINA

Questo è un programma in linguaggio di macchina o programma oggetto. Se si caricasse questo programma nella memoria di un microcomputer basato sullo Z80, questi potrebbe eseguirlo direttamente.

IL PROBLEMA DELLA PROGRAMMAZIONE

Esistono molte difficoltà legate alla creazione di programmi oggetto o programmi in linguaggio di macchina binario. Alcuni problemi sono questi:

1) I programmi sono difficili alla comprensione ed al debugging (messa a punto) (i numeri binari sembrano gli stessi, in modo particolare dopo averli visti per alcune ore).
2) I programmi si caricano lentamente poichè occorre caricare un bit alla volta.
3) I programmi non descrivono il compito che si vuole fare svolgere al computer in un formato leggibile da un essere umano qualunque.
4) I programmi sono lunghi e faticosi da scrivere.
5) Il programmatore compie spesso errori di disattenzione molto difficili da trovare.

Per esempio, **la seguente versione del programma oggetto della addizione contiene un errore di un solo bit. Provate a trovarlo:**

```
00111010
01100000
00000000
01000111
01110010
01100001
00000000
10000000
00110010
01100010
00000000
```

Sebbene il computer tratti i numeri binari con facilità, la gente no. Questa trova i programmi binari lunghi, faticosi, originanti confusione e senza significato. Eventualmente un programmatore può cominciare a ricordare qualche codice binario, ma tale sforzo potrebbe essere speso più proficuamente.

USO DELL'OTTALE O DELL'ESADECIMALE

OTTALE O ESADECIMALE

Possiamo qualche volta migliorare la situazione scrivendo istruzioni usando numeri ottali oppure esadecimali, piuttosto che binari. In questo libro useremo numeri esadecimali perchè sono più corti e perchè sono lo standard per l'industria del microelaboratore. La Tabella 1-1 definisce i digit esadecimali e i loro equivalenti binari. **Il programma Z80 per sommare due numeri diventa ora:**

```
3A
60
00
47
3A
61
00
80
32
62
00
```

La versione esadecimale è, almeno, più breve da scrivere e non così faticosa da esaminare.

Qualche volta è più facile trovare gli errori in una sequenza di digit esadecimali. La versione errata del programma dell'addizione, nella forma esadecimale, diventa:

```
3A
60
00
47
72
61
00
80
32
62
00
```

L'errore è più facile da individuare.

Che cosa ne facciamo di questo programma esadecimale? Il microelaboratore capisce solo codici d'istruzione binari. La risposta è che dobbiamo convertire i numeri esadecimali in numeri binari. Questa conversione è un compito ripetitivo e faticoso. La gente che ci prova, compie tutti i tipi di piccoli errori, quali vedere la linea sbagliata, omettere un bit o trasporre un bit o un digit.

CARICATORE ESADECIMALE

Questo compito ripetitivo e massacrante è, tuttavia, un lavoro perfetto per un computer. Questi non si stanca e non si annoia mai e non compie mai errori stupidi. **L'idea, quindi, è di scrivere un programma che prende i numeri esadecimali e li converte in numeri binari. Questo è un programma standard in dotazione a molti microelaboratori; viene denominato «caricatore (loader) esadecimale».**

Vale la pena avere un caricatore esadecimale? Se avete intenzione di scrivere un programma usando numeri binari e se siete disposti a caricare il programma nel computer nella sua forma binaria, non avrete bisogno del caricatore esadecimale.

Se avete scelto il caricatore esadecimale, dovrete pagare un prezzo per esso. Il caricatore esadecimale è esso stesso un programma che dovete caricare in memoria. Inoltre, il caricatore esadecimale occuperà memoria — memoria che potreste usare in qualche altro modo.

Il compromesso fondamentale, perciò, sono il costo e le esigenze di memoria nei confronti dei risparmi di tempo del programmatore.

Un caricatore esadecimale vale ben la pena del suo piccolo costo.

Un caricatore esadecimale non risolve certamente ogni problema di programmazione. La versione esadecimale del programma è ancora difficile da leggere o da capire; per esempio, essa non distingue le istruzioni dei dati o degli indirizzi, nè il listing del programma fornisce alcun suggerimento su che cosa fa il programma. Che cosa significano 32 o 47 o 3A? Il memorizzare una scheda piena di codici è difficilmente una proposta appetitosa. Inoltre, i codici saranno caricati in maniera diversa per differenti microelaboratori e il programma richiederà una grande quantità di documentazione.

Tabella 1-1. Tabella di Conversione Esadecimale

Digit Esadecimale	Equivalente Binario	Equivalente Decimale
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

MNEMONICA DEI CODICI D'ISTRUZIONE

Un'ovvia migliorìa nella programmazione è assegnare un nome ad ogni codice d'istruzione. Questa assegnazione di nomi ai codici di istruzione è chiamata «mnemonica» o «jogger» di memoria. La mnemonica delle istruzioni dovrebbe descrivere in qualche modo ciò che fa l'istruzione.

PROBLEMI DI MNEMONICA

Infatti ogni produttore di microelaboratori (non potrebbe ricordarsi neppure i codici esadecimali) fornisce una mnemonica di riferimento per l'insieme d'istruzioni del microelaboratore.
Non dovete essere fedeli alla mnemonica del produttore; non è sacra neppure per lui. Tuttavia essa è standard per un dato microelaboratore e quindi compresa da tutti gli utilizzatori. Questi sono i nomi di istruzioni che troverete in manuali, in schede, in libri, in articoli e in programmi. Il problema nella scelta di una mnemonica per le istruzioni consiste nel fatto che non tutte le istruzioni hanno nomi «ovvi». Alcune istruzioni hanno nomi ovvi (per esempio ADD, AND, OR), altre hanno ovvie forme contratte (per esempio SUB per sottrazione, XOR per OR-esclusivo), mentre altre ancora non hanno né l'una né l'altra cosa. Il risultato sono forme mnemoniche come WMP, PCHL e anche SOB (provate a immaginare cosa significa!). La maggior parte dei produttori perviene a nomi in gran parte ragionevoli e ad alcuni disperati. Tuttavia, sembra che gli utilizzatori che progettano la propria mnemonica raramente facciano molto meglio del produttore.

Insieme con la mnemonica delle istruzioni, il produttore assegnerà abitualmente nomi ai registri della CPU. Così come per i nomi delle istruzioni, alcuni nomi di registri sono evidenti (per esempio A per l'Accumulatore), mentre altri nomi possono avere solo un significato storico. Useremo i suggerimenti del produttore semplicemente per promuovere una standardizzazione.

PROGRAMMA IN LINGUAGGIO ASSEMBLY

Se usiamo la mnemonica delle istruzioni standard dello Z80 e dei registri, come definita dalla Zilog, il nostro programma dell'addizione con lo Z80 diventa:

```
LD      A,(60 H)
LD      B,A
LD      A,(61H)
ADD     A,B
LD      (62 H),A
```

Il programma è ancora lontano dall'essere evidente, ma almeno alcune parti sono comprensibili. ADD A,B è un miglioramento considerevole rispetto a 80; LD dà l'idea del caricamento di un dato in un registro o in una locazione di memoria. **Un programma siffatto è un programma in lingaggio assembly.**

IL PROGRAMMA ASSEMBLATORE

ASSEMBLAGGIO MANUALE

Come facciamo a mettere il programma in linguaggio assembly nel computer? Dobbiamo tradurlo o in numeri esadecimali o in numeri binari. **Si può tradurre un programma in linguaggio assembly manualmente,** istruzione per istruzione. Ciò è denominato assemblaggio manuale.

Un assemblaggio manuale dei codici delle istruzioni del programma di addizione si può illustrare come segue:

Nome dell'Istruzione	Equivalente Esadecimale
LD A,(NN)	3A
LD B,A	47
ADD A,B	80
LD (NN),A	32

Come nel caso della conversione da esadecimale a binario, un assemblaggio manuale è un lavoro meccanico non interessante, ripetitivo e soggetto a numerosi errori di poca importanza. Scegliere una linea sbagliata, trasporre digit, omettere istruzioni e leggere male i codici sono solo alcuni degli errori che si possono compiere. La maggior parte dei microelaboratori complicano il proprio lavoro fino ad avere istruzioni con diverse lunghezze di parola. Alcune istruzioni sono lunghe una sola parola mentre altre sono lunghe due o tre parole. Alcune istruzioni richiedono dati nella seconda e nella terza parola; altre richiedono indirizzi di memoria, numeri di registri o chi sa che cosa?

ASSEMBLATORE
PROGRAMMA SORGENTE
PROGRAMMA OGGETTO

L'assemblaggio è un altro lavoro meccanico che possiamo assegnare al microcomputer. Il microcomputer non fa mai errori quando traduce codici; esso conosce sempre quante parole e quale formato richiede ogni istruzione. Il programma che compie questo lavoro è denominato «assemblatore». Il programma assemblatore traduce un programma di un utilizzatore, o programma «sorgente» scritto secondo una mnemonica, in un programma in linguaggio macchina, o programma «oggetto», che il microcomputer può eseguire. L'ingresso dell'assemblatore è un programma sorgente e la sua uscita è un programma oggetto.

I compromessi dei quali abbiamo discusso in coincidenza con il caricatore esadecimale sono amplificati nel caso dell'assemblatore. Gli assemblatori sono più costosi, occupano più memoria e richiedono più periferiche e maggior tempo di esecuzione dei caricatori esadecimali. Mentre gli utilizzatori possono (e spesso lo fanno) scrivere i propri caricatori, pochi tengono a scrivere i propri assemblatori.

Gli assemblatori hanno regole proprie che dovete imparare a rispettare. Queste comprendono l'uso di certi segni (quali spazi, virgole, punti e virgola o due punti) in opportune posizioni, di una ortografia corretta, delle opportune informazioni di controllo, e forse anche del corretto posizionamento di nomi e di numeri. Queste regole sono tipicamente un ostacolo di minore importanza che può essere superato velocemente.

CARATTERISTICHE ADDIZIONALI DEGLI ASSEMBLATORI

I primi programmi assemblatori non facevano poco più che tradurre i nomi mnemonici delle istruzioni e dei registri nei loro equivalenti binari. Tuttavia ora la maggior parte degli assemblatori forniscono caratteristiche supplementari quali:

1) Permettere all'utilizzatore di assegnare nomi e locazioni di memoria, a dispositivi d'ingresso e di uscita, ed anche a sequenze di istruzioni.
2) Convertire dati od indirizzi da vari sistemi numerici (per esempio decimale o esadecimale) a binario e convertire caratteri nei loro codici binari ASCII o EBCDIC.
3) Effettuare alcune operazioni aritmetiche come parte di un processo di assemblaggio.
4) Dire al programma caricatore in quale zona di memoria porre parti del programma o dati.
5) Permette all'utilizzatore di assegnare zone di memoria per immagazzinamento temporaneo di dati e disporre dati fissi in zone di memoria di programma.
6) Fornire le informazioni necessarie per inserire nel programma corrente programmi standard da librerie di programmi o programmi scritti in altre occasioni.
7) Permettere all'utilizzatore di controllare il formato del listing del programma ed i dispositivi d'ingresso e d'uscita impiegati.

SCELTA DI UN ASSEMBLATORE

Tutte queste caratteristiche, naturalmente, comportano costi e memoria supplementari. Generalmente i microcomputer hanno assemblatori più semplici di quelli che hanno i computer più grandi, ma le dimensioni degli assemblatori tendono sempre a crescere. Spesso avrete possibilità di scelte di assemblatori. Il concetto importante non è quante caratteristiche fuori dell'usuale abbia un assemblatore, ma piuttosto quale convenienza offra nel lavoro di pratica normale.

SVANTAGGI DEL LINGUAGGIO ASSEMBLY

L'assemblatore, come il caricatore esadecimale, non risolve tutti i problemi della programmazione. Un problema è il tremendo divario tra l'insieme di istruzioni del microcomputer e i compiti che deve compiere il microcomputer. Le istruzioni del microcomputer tendono a fare cose come sommare il contenuto di due registri, fare scorrere il contenuto dell'Accumulatore di un bit o porre un nuovo valore nel Contatore di Programma. D'altra parte, un utilizzatore generalmente vuole che un microcomputer faccia qualcosa come verificare se un lettore analogico ha superato una soglia, rimanere in attesa e di reagire ad un particolare comando da una telescrivente o attivare un relé all'istante opportuno. Un programmatore in linguaggio assembly deve tradurre tali compiti in una sequenza di semplici istruzioni di un computer. La traduzione può essere un lavoro difficile con consumo di tempo.

Inoltre, **se state programmando in linguaggio assembly, dovete avere una conoscenza particolareggiata del microcomputer che state usando.** Dovete sapere quali registri e quali istruzioni ha il microcomputer, sapere con esattezza come le istruzioni influenzano i vari registri, quali metodi di indirizzamento usa il computer ed una infinità di altre informazioni. Nessuna di queste informazioni è attinente al lavoro che il microcomputer deve in definitiva svolgere.

PORTABILITÀ

Inoltre i programmi in linguaggio assembly non sono portabili. Ogni microcomputer ha il proprio linguaggio assembly, che rispecchia la propria architettura. Un programma in linguaggio assembly scritto per lo Z80 non potrà essere eseguito sul 6800 della Motorola, sull'F8 della Fairchild o sul PACE della National Semiconductor. Per esempio il programma di somma scritto per il 6800 della Motorola potrebbe essere:

```
LDAA   $60
ADDA   $61
STAA   $62
```

La mancanza di portabilità non solo significa che non potete usare il vostro programma in linguaggio assembly su un altro microcomputer, ma pure che non potete usare alcun programma che non sia stato specificatamente scritto per il microcomputer che state usando. Questo è un tipico svantaggio per i microcomputer, poiché questi dispositivi sono nuovi e per essi esistono pochi programmi in linguaggio assembly. Il risultato è che, troppo frequentemente voi rimanete sul vostro. Se avete bisogno di un programma per compiere un particolare compito, non lo troverete facilmente nelle piccole librerie di programmi che forniscono la maggior parte dei costruttori. Probabilmente non lo troverete né in un archivio, né in un articolo di un giornale né in qualche file di un vecchio programma. Probabilmente dovrete scriverlo voi stessi.

LINGUAGGI AD ALTO LIVELLO

COMPILATORE

La soluzione alle molte difficoltà associate ai programmi in linguaggio assembly è di usare linguaggi «ad alto livello» o «orientati alla procedura».
Tali linguaggi permettono di descrivere compiti in forme orientate ai problemi piuttosto che orientati al computer. Ogni statement (istruzione) in un linguaggio ad alto livello realizza una funzione riconoscibile; generalmente esso corrisponderà a molte istruzioni in linguaggio assembly. Un programma chiamato compilatore traduce il programma sorgente ad alto livello in istruzioni in codice oggetto o in linguaggio macchina.

FORTRAN

Esistono molti differenti linguaggi ad alto livello per diversi tipi di problemi. Se per esempio si può esprimere con notazione algebrica ciò che si vuole venga fatto dal computer, si può scrivere il programma in FORTRAN (Formula Translation Language), il più vecchio e uno dei più largamente usati linguaggi ad alto livello. Se, ora, si vuole sommare due numeri, si deve dire al computer:

```
SUM = NUMB1 + NUMB2
```

Questo è un programma molto più semplice (e molto più breve) sia di quello equivalente in linguaggio macchina che di quello equivalente in linguaggio assembly. Altri programmi ad alto livello comprendono il COBOL (per applicazioni commerciali), il PASCAL (un altro linguaggio algebrico), il PL/1 (una combinazione di FORTRAN, ALGOL e COBOL), l'APL ed il BASIC (linguaggi che sono noti in sistemi a time-sharing).

VANTAGGI DEI LINGUAGGI AD ALTO LIVELLO

I linguaggi ad alto livello rendono i programmi più facili e più veloci da scrivere. Una valutazione comune è che un programmatore può scrivere un programma circa dieci volte più velocemente in un linguaggio ad alto livello se paragonato con il linguaggio assembly. Ciò solo scrivendo il programma; non si conta la definizione del problema, la progettazione del programma, il debugging, il collaudo o la documentazione; tutte attività che diventano più semplici e più veloci. Il programma in linguaggio ad alto livello, per esempio, si documenta parzialmente da solo. Anche se non conoscete il FORTRAN, probabilmente sareste in grado di dire ciò che fa lo statement illustrato in precedenza.

INDIPENDENZA DALLA MACCHINA DEI LINGUAGGI AD ALTO LIVELLO

I linguaggi ad alto livello risolvono molti altri problemi relativi alla programmazione in linguaggio assembly. Il linguaggio ad alto livello ha la propria sintassi (usualmente definita da una standardizzazione nazionale o internazionale). Il linguaggio non fa riferimento all'insieme di istruzioni, che registri o ad altre caratteristiche di un particolare computer. Il compilatore fa attenzione a tutti questi particolari. I programmatori si possono concentrare solo sui propri compiti; non hanno bisogno di una comprensione particolareggiata dell'architettura della CPU in esame — a tale proposito essi non hanno bisogno di conoscere nulla sul computer per il quale stanno programmando.

PORTABILITÀ DEI LINGUAGGI AD ALTO LIVELLO

I programmi scritti in un linguaggio ad alto livello sono «portabili» — almeno in teoria. Essi possono essere eseguiti su un computer o su un microcomputer avente un compilatore standard per quel linguaggio. Contemporaneamente tutti i precedenti programmi scritti in un linguaggio ad alto livello per computer precedenti sono disponibili per la programmazione su un nuovo computer. Ciò può significare migliaia di programmi nel caso di un linguaggio comune come il FORTRAN o il BASIC.

SVANTAGGI DEI LINGUAGGI AD ALTO LIVELLO

Se tutte le belle cose dette sui linguaggi ad alto livello sono vere, se potete scrivere programmi più velocemente e renderli anche portabili, perché annoiarsi con linguaggi assembly? Chi vorrebbe preoccuparsi dei registri, dei codici delle istruzioni, della mnemonica e di tutte quelle cose di poco conto! Come al solito ci sono degli svantaggi che bilanciano i vantaggi.

SINTASSI DEI LINGUAGGI AD ALTO LIVELLO

Un ovvio problema è **che dovete imparare «le regole» o «le sintassi» di un qualunque linguaggio ad alto livello** che vogliate usare. Un linguaggio ad alto livello ha un insieme di regole abbastanza complicato. Troverete che ci vorrà molto tempo solo per ottenere un programma corretto come sintassi (e probabilmente anche allora esso non farà ciò che volevate). Un linguaggio ad alto livello per computer è come una lingua straniera. Se avete un po' di talento, vi abituerete alle regole e sarete capaci di produrre programmi che il compilatore accetterà. Imparare le regole e cercare di rendere il programma accettabile da un compilatore non contribuisce direttamente a compiere il vostro lavoro.

Ecco, per esempio, alcune regole di FORTRAN:

- Le label devono essere numeri posti nelle prime cinque colonne della scheda.
- Gli statement devono cominciare nella settima colonna.
- Le variabili intere devono cominciare con le lettere I, J, K, L, M o N.

COSTO DEI COMPILATORI

Un altro ovvio problema è **la necessità di un compilatore per tradurre programmi scritti in un linguaggio ad alto livello.** I compilatori sono costosi e usano una grande quantità di memoria. Mentre la gran parte degli assemblatori occupano da 2K a 16K byte di memoria (1K = 1024), i compilatori occupano da 4K a 64K byte di memoria. In tale modo la quantità di spese generali coinvolte nell'uso del compilatore è pittosto grande.

NOTAZIONE ALGEBRICA

Inoltre **solo alcuni compilatori semplificheranno l'implementazione del vostro problema.** Il FORTRAN, per esempio, si adatta bene a problemi che si possono esprimere con formule algebriche. Se, tuttavia, il vostro problema è di controllare una stampante, di stampare una stringa di caratteri o di controllare un sistema di allarme, esso non si può esprimere facilmente con notazione algebrica. Infatti la formulazione della soluzione con notazione algebrica può essere più goffa e più difficile della formulazione in linguaggio assembly. Una risposta è di usare un linguaggio ad alto livello più adatto. Esistono alcuni di tali linguaggi, ma essi sono molto meno usati e standardizzati del FORTRAN. Non otterrete molti dei vantaggi dei linguaggi ad alto livello se usate questi linguaggi chiamati «linguaggi ad implementazione di sistema».

INEFFICIENZA DEI LINGUAGGI AD ALTO LIVELLO

I linguaggi ad alto livello non producono programmi in linguaggio di macchina molto efficienti. Il motivo fondamentale di ciò è che la compilazione è un processo automatico che è pieno di compromessi per permettere molti campi di possibilità. Il compilatore lavora in modo molto simile ad un traduttore di linguaggio di computer — qualche volta le parole sono giuste ma i suoni e le strutture delle frasi sono sgraziati. Un semplice compilatore non può sapere quando una variabile non è più usata e può essere scartata, o quando si potrebbe usare un registro piuttosto di una locazione di memoria o quando delle variabili hanno dei semplici rapporti. Il programmatore con esperienza può trarre vantaggio da semplificazioni per abbreviare il tempo di esecuzione o per ridurre l'uso di memoria. Alcuni compilatori (noti come compilatori ottimizzanti) possono fare ciò, ma essi sono più grandi e più lenti dei compilatori regolari.

I vantaggi e gli svantaggi generali dei linguaggi ad alto livello sono:

VANTAGGI DEI LINGUAGGI AD ALTO LIVELLO

Vantaggi:

- Descrizioni più convenienti dei compiti.
- Codificazione dei programmi più efficiente.
- Documentazione più facile.
- Sintassi standard.
- Indipendenza dalla struttura di un particolare computer.
- Portabilità.
- Disponibilità di libreria e di altri programmi.

SVANTAGGI DEI LINGUAGGI AD ALTO LIVELLO

Svantaggi:

- Regole speciali.
- Necessità di hardware costoso e di supporto per il software.
- Orientamento di linguaggi comuni verso problemi algebrici o commerciali.
- Programmi inefficienti.
- Difficoltà ad ottimizzare codici per soddisfare requisiti di tempo e di memoria.
- Inabilità ad usare convenientemente speciali caratteristiche di un computer.

LINGUAGGI AD ALTO LIVELLO PER MICROELABORATORI

Gli utilizzatori di microprocessori incontreranno parecchie e speciali difficoltà nell'usare linguaggi ad alto livello. Tra di queste si trovano le seguenti:

- Esistono pochi linguaggi ad alto livello per microelaboratori.
- Non è largamente disponibile nessun linguaggio ad alto livello.

- Pochi compilatori vengono eseguiti realmente su microelaboratori. Quelli che lo fanno richiedono una quantità di memoria molto grande.
- La gran parte delle applicazioni dei microelaboratori non si adattano molto bene ai linguaggi ad alto livello.
- I costi di memoria spesso sono critici in applicàzioni di microelaboratori.

La mancanza di linguaggi ad alto livello dipende parzialmente dal fatto che i microelaboratori sono alquanto nuovi e sono prodotti di costruttori di semiconduttori piuttosto che di costruttori di computer.

Esistono pochissimi linguaggi ad alto livello per microelaboratori. I più comuni sono i linguaggi tipo il PL/1 (quali il PL/M dell'Intel, l'MPL della Motorola e il PLμS della SIGNETICS), il BASIC e il PASCAL.

Inoltre i pochi linguaggi ad alto livello che esistono non sono conformi a standardizzazioni riconosciute, cosicché l'utilizzatore di un microelaboratore non può aspettarsi di guadagnare molta portabilità dei programmi, di accedere a librerie di programmi, o di usare programmi o esperienza precedenti. I principali vantaggi che rimangono sono la riduzione dello sforzo di programmazione e la minor quantità di conoscenza particolareggiata dell'architettura del computer necessaria.

SPESE GENERALI PER LINGUAGGI AD ALTO LIVELLO

Le spese generali coinvolte nell'uso di un linguaggio ad alto livello con i microelaboratori sono considerevoli. Gli stessi microelaboratori si adattano meglio a controllare e a rallentare le applicazioni interattive che alla manipolazione dei caratteri e all'analisi del linguaggio implicata nella compilazione. Perciò, gran parte dei compilatori per microelaboratori non verrà eseguita su un sistema basato su un microelaboratore. Invece essi richiedono un computer molto più grande, cioè, essi sono cross-compilatori piuttosto che auto-compilatori. Un utilizzatore deve non solo sopportare le spese di un computer più grande ma deve pure trasferire fisicamente il programma dal computer più grande al microelaboratore.

INADATTABILITÀ DEI LINGUAGGI AD ALTO LIVELLO

Sono disponibili alcuni auto-compilatori. Questi vengono eseguiti sul microcomputer per il quale producono il codice oggetto. Sfortunatamente, essi richiedono grandi quantità di memoria (16K o più), più hardware e software speciali di supporto. Inoltre i linguaggi ad alto livello generalmente non si adattano bene ad applicazioni con microelaboratori. Gran parte dei linguaggi comuni sono stati progettati o per aiutare a risolvere problemi scientifici o per gestire elaborazioni di dati commerciali in grande scala. Poche applicazioni dei microelaboratori rientrano in una di queste aree. Gran parte delle applicazioni con microelaboratori coinvolge la trasmissione di dati e le informazioni di controllo verso dispositivi di uscita e la ricezione di dati e le informazioni di stato da dispositivi d'ingresso. Spesso le informazioni di controllo e di stato consistono in alcuni digit binari con ben precisi significati relativi all'hardware. Se cercate di scrivere un tipico programma di controllo in un linguaggio ad alto livello, vi sentirete spesso come uno che sta cercando di mangiare la zuppa con due bacchette d'avorio. Per problemi relativi a equipaggiamenti di test, terminali, sistemi di navigazione, elaborazione di segnali ed equipaggiamenti commerciali, i linguaggi ad alto livello funzionano meglio di quanto non facciano nel campo della strumentazione, delle comunicazioni, delle periferiche e delle applicazioni automobilistiche.

AREE DI APPLICAZIONE PER LIVELLI DI LINGUAGGIO

Applicazioni che meglio si adattano a linguaggi ad alto livello sono quelle che richiedono grandi memorie. Se, come in un controllore a valvola, in un gioco elettronico, in un controllore di strumento, o in un piccolo strumento, il costo di un solo chip di memoria è importante, allora l'inefficienza dei linguaggi ad alto livello è intollerabile. Se, d'altra parte, come in un terminale o in un equipaggiamento di test, il sistema ha in ogni caso molte migliaia di byte di memoria, l'inefficienza dei linguaggi ad alto livello non è così importante. Chiaramente

pure le dimensioni del programma e il volume del prodotto sono fattori importanti. Un programma esteso aumenterà notevolmente i vantaggi dei linguaggi ad alto livello. D'altra parte, un'applicazione di grande volume significherà che i costi fissi di sviluppo del software non sono così importanti come i costi di memoria che sono parte costituente ogni sistema.

CHE LIVELLO DOVRETE USARE?

Dipende dalla vostra particolare applicazione. Prendiamo brevemente nota di alcuni fattori che possono favorire particolari livelli:

Linguaggio di Macchina:

APPLICAZIONI PER LINGUAGGIO DI MACCHINA

- **Virtualmente nessuno programma in linguaggio di macchina. Il suo uso non può essere giustificato considerando il basso costo di un assemblatore e l'aumento di velocità di programmazione che dà un assemblatore.**

Linguaggio Assembly:

APPLICAZIONI PER LINGUAGGIO ASSEMBLY

- **Programmi da brevi a moderate dimensioni.**
- **Applicazioni dove un fattore importante è il costo di memoria.**
- **Applicazioni di controllo in tempo reale.**
- **Elaborazioni di dati limitati.**
- **Applicazioni di grande volume.**
- **Più ingresso/uscita o controllo che calcolo.**

Linguaggi ad alto livello:

APPLICAZIONI PER LINGUAGGIO AD ALTO LIVELLO

- **Programmi lunghi.**
- **Applicazioni di basso volume richiedenti programmi lunghi.**
- **Applicazioni richiedenti grandi memorie.**
- **Più calcolo che ingresso/uscita o controllo.**
- **Compatibilità con applicazioni similari utilizzando computer più grandi.**
- **Disponibilità di programmi specifici in un linguaggio ad alto livello che può essere usato nell'applicazione.**

Sono pure importanti altri fattori, come la disponibilità di un computer più grande da usare nello sviluppo, l'esperienza con particolari linguaggi e la compatibilità con altre applicazioni.

Se in definitiva l'hardware sarà il maggior costo nella vostra applicazione o se la velocità è critica, dovrete dare la preferenza ad un linguaggio assembly. Ma preparatevi a consumare tempo supplementare nello sviluppo del software in cambio di minori costi di memoria e di maggiori velocità di esecuzione. Se il software sarà il maggior costo nella vostra applicazione, dovrete dare la preferenza ad un linguaggio ad alto livello. Ma preparatevi a consumare la memoria in più richiesta per sopportare l'hardware e il software.

Naturalmente, nessuno tranne qualche teorico avrà nulla da obiettare se userete sia linguaggio assembly che linguaggio ad alto livello. Potete in origine scrivere il programma in un linguaggio ad alto livello e quindi raffazzonare alcune sezioni in linguaggio assembly. Tuttavia, gran parte degli utilizzatori preferisce non fare ciò a causa della devastazione che si creerebbe nel debugging, nel collaudo e nella documentazione.

E PER IL FUTURO?

Ci aspettiamo che il futuro tenderà a favorire il linguaggio ad alto livello per i seguenti motivi:

ORIENTAMENTI FUTURI NEI LIVELLI DI LINGUAGGIO

- Sembra che si aggiungano sempre caratteristiche in più ai programmi e che questi diventino più estesi
- L'hardware e le memorie stanno diventando meno costosi
- Il software e i programmatori stanno diventando più costosi
- I chip di memoria stanno diventando disponibili in dimensioni maggiori, ad un costo «per bit» minore, così che il risparmio reale nei chip è meno probabile.
- Stanno diventando disponibili più compilatori.
- Si stanno sviluppando linguaggi ad alto livello più adattabili e più efficienti.
- Si avrà una maggiore standardizzazione dei linguaggi ad alto livello.

La programmazione in linguaggio assembly dei microelaboratori sarà un'arte morente non più di quanto lo sia ora per i grandi computer. Ma programmi più lunghi, memorie più economiche, e programmatori più costosi renderanno i costi del software la parte maggiore di gran parte delle applicazioni. In molte applicazioni perciò ci si avvierà verso linguaggi ad alto livello.

PERCHÈ QUESTO LIBRO?

Se il futuro sembra favorire i linguaggi ad alto livello, perchè avere un libro sulla programmazione in linguaggio assembly? Le ragioni sono:

1) Gran parte degli utilizzatori dei microcomputer correnti programmano in linguaggio assembly (quasi 2/3, secondo una indagine recente).
2) Molti utilizzatori di microcomputer continueranno a programmare in linguaggio assembly poichè hanno bisogno del controllo dettagliato che questi fornisce.
3) Non sono ancora largamente disponibili o standardizzati linguaggi ad alto livello convenienti.
4) Molte applicazioni richiedono l'efficienza del linguaggio assembly.
5) Una conoscenza del linguaggio assembly può aiutare nella valutazione dei linguaggi ad alto livello.

La parte restante di questo libro tratterà esclusivamente degli assemblatori e della programmazione in linguaggio/assembly. Tuttavia, vogliamo che i lettori sappiano che il linguaggio assembly non è la sola alternativa. Dovreste osservare i nuovi sviluppi che possono ridurre in modo significativo i costi di programmazione se tali costi sono il fattore di maggiore importanza nella vostra applicazione.

BIBLIOGRAFIA

Alcuni confronti globali sul tempo richiesto per scrivere diversi tipi di programmi in differenti livelli di linguaggio si trovano in M.H. Halstead, Elements of Software Science, American Elsevier, New York, 1977 e in V. Schneider, «Prediction of Software Effort and Project Duration — Four New Formulas» SIGPLAN Notices, June 1978, pp. 48-55.

Capitolo 2

ASSEMBLATORI

Questo capitolo tratta delle funzioni svolte dagli assemblatori, cominciando con caratteristiche comuni alla maggior parte degli assemblatori, e proseguendo con possibilità più elaborate quali i macro e l'assembly condizionale. Potete saltare questo capitolo, se lo desiderate, e tornarvi sopra quando vi sentirete più a vostro agio con l'argomento.

CARATTERISTICHE DEGLI ASSEMBLATORI

Come ricordato precedentemente, gli assemblatori odierni fanno molto più che tradurre la mnemonica del linguaggio assembly in codici binari. Ma dapprima descriveremo come un assemblatore gestisce la traduzione della mnemonica prima di descrivere le caratteristiche aggiuntive. Infine, spiegheremo come si usano gli assemblatori.

ISTRUZIONI DELL'ASSEMBLATORE

CAMPI DEL LINGUAGGIO ASSEMBLY

Le istruzioni in linguaggio assembly (o «statement») sono divise in un certo numero di «campi», come mostrato in Tabella 2-1.

Il campo del codice operativo è il solo campo che non può essere mai vuoto; contiene sempre o un codice mnemonico di istruzione o una direttiva per l'assemblatore, denominata pseudo-operazione o pseudo-op.

Il campo indirizzo può contenere un indirizzo o un dato o può essere vuoto.

Tabella 2-1. I Campi di un'Istruzione in Linguaggio Assembly

Campo di Label	Codice Operativo o Campo Mnemonico	Operando o Campo Indirizzo	Campo Commento
START:	LD	A,(VAL1)	;LOAD FIRST NUMBER INTO A
	LD	B,A	;SAVE IN B
	LD	A,(VAL2)	;LOAD SECOND NUMBER INTO A
	ADD	A,B	;ADD FIRST NUMBER TO A
	LD	(SUM),A	;STORE SUM
NEXT:	?	?	;NEXT INSTRUCTION
VAL1:	DEFS	1	
VAL2:	DEFS	1	
SUM:	DEFS	1	

I campi di commento e di label sono opzionali. Un programmatore assegnerà una label ad uno statement o aggiungerà un commento per convenienza personale, per es., per rendere il programma più facile da codificare e da leggere.

FORMATO

Naturalmente l'assemblatore deve avere qualche possibilità di dire dove finisce un campo o dove ne inizia un altro. L'assemblatore che usa un ingresso a carta perforata richiede spesso che ogni campo inizi in una colonna specifica della scheda. Questo è un formato rigido. Tuttavia, i formati rigidi possono essere non convenienti quando lo strumento d'ingresso è un nastro di carta; i formati rigidi sono pure fastidiosi per i programmatori. L'alternativa è un formato libero, dove i campi possono essere dovunque sulle linee.

DELIMITATORI

Se l'assemblatore non può usare la posizione nella linea per dire i campi in disparte, deve usare qualche altra cosa. **Gran parte degli assemblatori usano un simbolo speciale o delimitatore all'inizio o alla fine di ogni campo.** Il delimitatore più banale è il carattere di spazio. Virgole, punto, punti e virgole, due punti, trattini, punti interrogativi ed altri caratteri che non sarebbero altrimenti usati nei programmi in linguaggio assembly possono pure servire come delimitatori. La Tabella 2-2 elenca delimitatori standard dell'assemblatore dello Z80 della Zilog.

Tabella 2-2. Delimitatori Standard dell'Assemblatore Z80

:	dopo una label
'spazio'	tra un codice operativo ed un indirizzo
,	tra operandi nel campo indirizzo
;	prima di un commento

Dovreste esercitarvi con cura sui delimitatori. Qualche assemblatore dà troppa importanza a spazi in più o alla comparsa di delimitatori nei commenti o nelle label. Un assemblatore ben formulato maneggierà questi problemi di minore importanza, ma molti assemblatori non sono scritti bene. La nostra raccomandazione è semplice: evitare, se possibile, problemi potenziali. Le seguenti regole vi aiuteranno:

1) Non usare spazi in più, in particolare dopo virgole che separano operandi.
2) Non usare caratteri delimitatori in nomi o in label.
3) Inserire delimitatori standard anche se il vostro assemblatore non li richiede. I vostri programmi si potranno allora assemblare con un qualunque assemblatore.

LABEL

CAMPO DI LABEL

Il campo di label è il primo campo in un'istruzione in linguaggio assembly; può essere uno spazio. Se una label è presente, l'assemblatore assegna alla label il valore dell'indirizzo della locazione di memoria nella quale è stato caricato il primo byte di programma oggetto relativo a quell'istruzione. Successivamente potete usare la label come dato o con un indirizzo in un campo operando di un'altra istruzione. L'assemblatore sostituirà la label col valore assegnato quando costruisce un programma oggetto.

LABEL NELLE ISTRUZIONI DI SALTO

Le label sono usate di frequente nelle istruzioni di Jump, di Call o di Branch. Queste istruzioni mettono un nuovo valore nel Contatore di Programma, alterando così la normale esecuzione delle istruzioni. JUMP 150_{16} significa «metti il valore 150_{16} nel Contatore di Programma». La successiva istruzione da eseguire sarà quella contenuta nella locazione di memoria 150_{16}. L'istruzione JUMP START significa «metti nel Contatore di Programma il valore assegnato alla label START». La successiva istruzione da eseguire sarà quella contenuta nella locazione di memoria alla quale è stata assegnata la label START. La Tabella 2-3 contiene un esempio.

Tabella 2-3. Assegnazione ed Utilizzazione di una Label

```
                    ASSEMBLY LANGUAGE PROGRAM

START               LOAD ACCUMULATOR 100
                    .
                    .
                    . (MAIN PROGRAM)
                    .
                    .
                    .
                    JUMP START
```

Quando si esegue la versione in linguaggio macchina di questo programma, l'istruzione JUMP START farà sì che nel Contatore di Programma venga messo l'indirizzo dell'istruzione con label START. Successivamente verrà eseguita l'istruzione con label START.

Perchè usare una label? Ecco alcuni motivi:

1) Una label rende un programma più facile da trovare e da ricordare.
2) La label può essere spostata per modificare o correggere un programma. Non dovete cambiare alcuna istruzione successiva che usa la label: l'assemblatore farà tutte le modifiche necessarie.

COSTANTE DI RIALLOCAZIONE

3) L'assemblatore o il caricatore può riposizionare l'intero programma aggiungendo una costante (una costante di «riallocazione») ad ogni indirizzo in cui si usa una label. In tale modo si può spostare il programma per permettere l'inserimento di altri programmi semplicemente per riordinare la memoria.
4) Il programma è più facile da usare come programma di libreria, cioè è più facile per qualcun altro prendere il vostro programma ed aggiungerlo a qualche programma totalmente diverso.
5) Non dovete calcolare indirizzi di memoria. Il calcolo di indirizzi di memoria è particolarmente difficile con i microelaboratori i quali hanno istruzioni che variano come lunghezza.

Fa sensazione assegnare ad un'istruzione una label che si può voler usare come destinazione o identificare altrimenti.

SCELTA DI LABEL

La prossima domanda è quale label usare. Spesso l'assemblatore pone delle restrizioni sul numero di caratteri (di solito 5 o 6), sul carattere iniziale (spesso deve essere una lettera) e sui caratteri finali (spesso devono essere lettere, numeri o uno dei pochi caratteri speciali). Al di là di queste restrizioni, la scelta spetta a voi.

La vostra preferenza **va all'uso di label che suggeriscono il loro scopo,** cioè, label mnemoniche. Esempi tipici sono ADDW in una routine che somma una parola ad una somma, SRETX in una routine che fa la ricerca del carattere in ASCII ETX o NKEYS per una locazione nella memoria dati che contiene il numero di ingressi a chiave. Label significative sono più facili da ricordare e contribuiscono alla documentazione del programma. Alcuni programmatori preferiscono usare un formato standard per le label, come iniziare con L0000. Queste label succedono l'una dopo altre (potete fare scorrere alcuni numeri per permettere inserimenti), ma non aiutano a documentare il programma.

REGOLE PER SCEGLIERE LABEL

Alcune regole di scelta della label vi terranno fuori dai guai. Raccomandiamo le seguenti:

1) Non usare label che sono le stesse sia come codici operativi che come altra mnemonica. Gran parte degli assemblatori non permetteranno questo impiego, che confonde molto.

2) Non usare label più lunghe di quanto permette l'assemblatore. Gli assemblatori hanno diverse regole di troncamento.
3) Evitare caratteri speciali (non alfabetici e non numerici) e lettere maiuscole. Alcuni assemblatori non lo consentono; altri ne permettono solo alcune. La consuetudine più semplice è di attenersi alle lettere maiuscole e ai numeri.
4) Iniziare label con una lettera. Label siffatte sono sempre accettabili.
5) Non usare label che potrebbero confondersi con altre. Evitare le lettere I, O e Z e i numeri 0, 1 e 2. Evitare pure cose come XXXX e XXXXX. Non c'è alcun senso nel mettere alla prova il destino e le leggi di Murphy («se una cosa può andare storta, andrà sicuramente storta»).
6) Quando non siete sicuri se una label è legittima, non usatela. Non ricaverete alcun beneficio reale nello scoprire con esattezza ciò che accetterà l'Assemblatore.

Queste sono raccomandazioni, non regole. Non dovete seguirle, ma non biasimateci se perderete tempo su problemi stupidi.

CODICI OPERATIVI DELL'ASSEMBLATORE (MNEMONICA)

Il compito principale dell'assemblatore è la traduzione dei codici operativi mnemonici nei loro equivalenti codici binari. L'assemblatore porta a termine questo compito usando una tabella fissa così come fareste se assemblaste a mano.

L'assemblatore deve, tuttavia, fare più che tradurre i codici operativi. **Deve pure, in qualche modo, determinare quanti operandi richiede l'istruzione e di che tipo sono.** Ciò può essere abbastanza complesso — alcune istruzioni (come un Halt) non hanno nessun operando, altre (come un Addition o un'istruzione Jump) ne hanno uno, mentre altre ancora (come un trasferimento tra registri o uno scorrimento multiplo di bit) ne richiedono due. Alcune istruzioni possono pure permettere alternative, per es. alcuni computer hanno istruzioni (con Shift o Clear) che si possono applicare sia all'Accumulatore che ad una locazione di memoria. Non tratteremo di come l'assemblatore faccia queste distinzioni; facciamo solo notare che può fare così.

PSEUDO - OPERAZIONI

PSEUDO-OPERAZIONI

DIRETTIVA DI ASSEMBLATORE

Alcune istruzioni in linguaggio assembly non sono tradotte direttamente in istruzioni in linguaggio di macchina. Queste istruzioni sono direttive di assemblatore; esse assegnano il programma a certe aree di memoria, definiscono simboli, stabiliscono zone di RAM per immagazzinamento temporaneo di dati, mettono in memoria tabelle o altri dati fissi, permettono riferimenti ad altri programmi e svolgono funzioni governo di minore importanza.

Per usare queste direttive di assemblatore o pseudo-istruzioni, un programmatore mette la mnemonica della pseudo-operazione nel campo del codice operativo e, se è richiesto dalla pseudo-operazione specificata, un indirizzo o un dato nel campo indirizzo.

Le pseudo-operazioni più comuni sono:

DATA
EQUATE o DEFINE
ORIGIN
RESERVE

Le pseudo-operazioni di collegamento sono:

ENTRY
EXTERNAL

Assemblatori differenti usano nomi diversi per queste operazioni, ma gli scopi sono gli stessi. Le pseudo-operazioni di governo interno comprendono:

END
LIST
NAME
PAGE
SPACE
TITLE

Tratteremo brevemente queste pseudo-istruzioni, anche se le loro funzioni sono di solito ovvie.

LA PSEUDO-OPERAZIONE DATA

La pseudo-operazione DATA permette al programmatore di mettere dati fissi in memoria. Questi dati possono comprendere:

- Tabelle di ricerca (o di consultazione).
- Tabelle di conversione di codici.
- Messaggi.
- Configurazioni di sincronizzazione.
- Soglie.
- Nomi.
- Coefficienti per equazioni.
- Comandi.
- Fattori di conversione.
- Fattori di peso.
- Tempi o frequenze caratteristici.
- Indirizzi di sottoprogrammi.
- Identificazioni di tasti.
- Configurazioni di test (test pattern).
- Configurazioni per generazione di caratteri.
- Configurazioni di identificazione.
- Tabelle di registrazione.
- Forme standard.
- Maschere.
- Tabelle di transizioni di stato.

La pseudo-istruzione DATA tratta il dato come parte permanente del programma.

Il formato di una istruzione DATA è di solito molto semplice. Un'istruzione come:

```
DZCON   DATA    12
```

metterà il numero 12 nella prossima locazione di memoria disponibile ed assegnerà a quella locazione il nome DZCON. Di solito ogni pseudo-istruzione DATA ha una label, a meno che faccia parte di una serie di pseudo-istruzioni DATA. Il dato e la label possono assumere qualunque forma consentita dall'assemblatore.

Gran parte degli assemblatori permettono istruzioni DATA più elaborate che gestiscono una grande quantità di dati contemporaneamente, per esempio:

```
EMESS   DATA    'ERROR'
SQRS    DATA    1, 4, 9, 16, 25
```

Una singola istruzione può riempire molte parole di memoria programmi, limitate solo dalla lunghezza di una linea. Si noti che se non si possono mettere tutti i dati su una sola linea, si può sempre far seguire una istruzione DATA con un'altra, per esempio:

```
MESSG   DATA    'ORA È IL'
        DATA    'TEMPO PER TUTTI'
        DATA    'GLI UOMINI BUONI'
        DATA    'DI VENIRE IN'
        DATA    'AIUTO AL LORO'
        DATA    'PAESE'
```

Assemblatori per microelaboratori hanno tipicamente alcune variazioni di pseudo-operazioni DATA standard. DEFINE BYTE o FORM CONSTANT BYTE gestiscono numeri ad 8 bit; DEFINE WORD o FORM CONSTANT WORD gestiscono numeri a 16 bit o indirizzi. Altre pseudo-operazioni speciali possono gestire dati codificati come caratteri.

LA PSEUDO-OPERAZIONE EQUATE (o DEFINE)

DEFINIZIONI DI NOMI

La pseudo-operazione EQUATE permette al programmatore di eguagliare label e nomi ad indirizzi e dati. Questa pseudo-operazione è quasi sempre data dalla mnemonica EQU. I nomi possono rifarsi ad indirizzi di dispositivi, a dati numerici, ad indirizzi di partenza, ad indirizzi fissi, etc.

La pseudo-operazione EQUATE assegna il valore numerico nel suo campo operando alla label nel suo campo label. Ecco due esempi:

```
TTY     EQU     5
LAST    EQU     5000
```

Gran parte degli assemblatori vi permetteranno di definire una label in termini di un'altra, per esempio:

```
LAST    EQU     FINAL
ST1     EQU     START + 1
```

La label nel campo operando deve, naturalmente, essere stata definita precedentemente. Spesso il campo operando può contenere espressioni più complesse, come vedremo dopo. Assegnazioni con doppio nome (due nomi per lo stesso dato o indirizzo) possono essere utili nel mettere insieme programmi che usano differenti nomi per la stessa variabile (o differente ortografia per ciò che si è ipotizzato essere lo stesso nome).

TABELLA DEI SIMBOLI

Si noti che una pseudo-operazione EQU fa sì che l'assemblatore non metta in memoria una cosa qualunque. L'assemblatore carica semplicemente un nome in una tabella (chiamata tabella dei simboli) che l'assemblatore conserva. Questa tabella, diversamente dalla tabella mnemonica, deve essere in RAM, poichè essa cambia per ogni programma. Il programma assemblatore avrà sempre bisogno di RAM per conservare la tabella dei simboli; più RAM ha, più simboli può accettare. Questa RAM è in aggiunta a quella di cui l'assemblatore ha bisogno per memorizzazioni temporanee.

USO DI NOMI

Quando si usa un nome? La risposta è: ogniqualvolta si ha un parametro che ha significato al di là del suo valore numerico ordinario o il valore numerico del parametro può essere cambiato. Tipicamente assegnamo nomi e costanti di tempo, ad indirizzi di dispositivi, a maschere, a fattori di conversione e simboli. Un nome come DELAY, TTY, KBD, NROW o OPEN non semplifica il cambiamento del parametro, ma documenta pure il programma. Assegnamo nomi anche a locazioni di memoria che hanno scopi speciali; possono contenere dati, indicare l'inizio del programma o essere disponibili per immagazzinamenti intermedi.

SCELTA DI NOMI

Che nome usare? Le regole migliori sono le stesse che nel caso delle label, tranne che qui realmente contano nomi significativi. Perchè non chiamare la telescrivente TTY invece di X15, un ritardo di tempo di un bit BTIME o BTDLY piuttosto che WW, il numero del tasto «GO» di una tastiera GOKEY, piuttosto che HORSE? Questo consiglio sembra giusto, ma un sorprendente numero di programmatori non lo segue.

POSIZIONAMENTO DELLE DEFINIZIONI

Dove si mettono le pseudo-operazioni EQUATE? Il posto migliore è all'inizio del programma, sotto appropriati commenti quali INDIRIZZI DI I/O, IMMAGAZZINAMENTO TEMPORANEO, COSTANTI DI TEMPO, o LOCAZIONI DI PROGRAMMA. Ciò rende le definizioni facili da trovare se volete cambiarle. Inoltre, un altro utilizzatore sarà in grado di ricercare tutte le definizioni in un solo posto centralizzato. Chiaramente questo metodo pratico migliora la documentazione e rende il programma più facile da usare.

Le definizioni usate solo in un sottoprogramma specifico dovrebbero comparire all'inizio del sottoprogramma.

LA PSEUDO-OPERAZIONE ORIGIN

La pseudo-operazione ORIGIN (quasi sempre abbreviata in ORG) permette al programmatore di allocare dovunque in memoria programmi, sottoprogrammi o dati. Programmi e dati possono essere allocati in differenti aree di memoria in funzione della configurazione di memoria. Routine di inizializzazione, routine di servizio delle interruzioni ed altri programmi richiesti possono essere distribuiti in memoria ad indirizzi fissi o convenienti.

CONTATORE DI LOCAZIONE

L'assemblatore gestisce un Contatore di Locazione (paragonabile al Contatore di Programma del computer) che contiene la locazione di memoria in cui risiederà il prossimo byte di codice oggetto generato dall'assemblatore quando sarà stato caricato il programma. Una pseudo-operazione ORG fa sì che l'assemblatore metta un nuovo valore nel Contatore di Locazione, come una istruzione Jump fa sì che la CPU metta un nuovo valore nel Contatore di Programma. L'uscita dell'assemblatore deve non solo contenere istruzioni o dati, ma pure indicare al programma caricatore dove si potrebbero mettere istruzioni e dati in memoria.

I programmi per microelaboratori spesso contengono parecchi statement ORG per i seguenti scopi:

Indirizzo di Reset (Partenza)	Programma Principale
Indirizzi di servizio degli interrupt	Sottoprogrammi
Indirizzi trappola	Indirizzi di memoria per
Immagazzinamento in RAM	dispositivi d'ingresso/uscita
Stack di memoria	o funzioni speciali

Altri statement ORIGIN possono inoltre consentire spazio per inserzioni successive, posizionare tabelle o dati in memoria o assegnare spazio di RAM libera per buffer di dati. Le memorie programmi e dati nei microcomputer possono occupare indirizzi molto sparsi per semplificare l'hardware.

Tipici statement ORIGIN sono:

```
ORG     RESET
ORG     1000
ORG     INT3
```

Alcuni assemblatori assumono un'origine pari a zero se il programmatore non mette uno statement ORG all'inizio del programma. La convenienza è trascurabile; raccomandiamo l'inserzione di uno statement ORG per evitare confusione.

LA PSEUDO-OPERAZIONE RESERVE

ALLOCAZIONE DI RAM

La pseudo-operazione RESERVE permette al programmatore di allocare RAM per vari scopi quali tabelle di dati, immagazzinamento temporaneo, indirizzi indiretti, uno Stack, etc.

Con l'uso della pseudo-operazione RESERVE si assegna un nome all'area di memoria e si dichiara il numero di locazioni che devono essere assegnate. Ecco alcuni esempi:

```
NOKEY   RESERVE   1
TEMP    RESERVE   50
VOLTG   RESERVE   80
BUFR    RESERVE   100
```

Potete usare la pseudo-operazione RESERVE per riservare locazioni di memoria in memoria di programma o in memoria dati; tuttavia la natura della pseudo-operazione RESERVE è più significativa quando è applicata alla memoria dati.

In realtà tutto ciò che fa la pseudo-istruzione RESERVE è di incrementare il Contatore di Locazione dell'assemblatore della quantità dichiarata nel campo operando. L'assemblatore non produce in realtà nessun codice oggetto.

Si notino le seguenti caratteristiche di RESERVE:

1) La label della pseudo-operazione RESERVE è assegnata al valore del primo indirizzo riservato. Per esempio, la sequenza:

```
        ORG       3000
BUF1    RESERVE   100
BUF2    RESERVE   50
VOLTS   RESERVE   5
```

   assegna alla label BUF1 il valore 3000, a BUF2 3100 e a VOLTS 3150.
2) Dovete specificare il numero di locazioni da riservare. Non c'è nessun caso di errore.
3) Nessun dato è messo nelle locazioni riservate. Qualunque dato che, per puro caso, si trovi in queste locazioni sarà lasciato lì.

INIZIALIZZAZIONE DELLA RAM

Alcuni assemblatori permettono al programma di mettere valori iniziali in RAM. Raccomandiamo fortemente di non usare questa caratteristica — si suppone che il programma (insieme con valori iniziali) sarà caricato da un dispositivo esterno (per es. nastro di carta o floppy disk) ogni volta che deve essere eseguito. Gran parte dei programmi per microprocessori, d'altra parte, risiedono in ROM non volatili e partono quando si dà tensione. La RAM in tali situazioni non mantiene il suo contenuto, né questi è ricaricato. Inserite sempre nel vostro programma istruzioni per inizializzare la RAM.

PSEUDO-OPERAZIONI DI LINKING

RIFERIMENTI ESTERNI

Spesso in un programma o in un sottoprogramma vogliamo degli statement per usare nomi definiti altrove. Tali nomi sono chiamati <u>riferimenti esterni;</u> è necessario uno speciale programma «linker» per inserire realmente valori esterni e determinare se qualche nome non è definito o definito due volte.

La pseudo-operazione EXTERNAL, di solito abbreviata in EXT, sta a significare che il nome è definito altrove.

La pseudo-operazione ENTRY, abbreviata di solito in ENT, sta a significare che il nome è disponibile da usare altrove, cioè, è definito in questo programma.

Il modo esatto di come il linking di pseudo-operazioni è implementato varia grandemente da assemblatore ad assemblatore. Non faremo più riferimento a tali **pseudo-operazioni,** ma esse **sono molto utili in applicazioni pratiche.**

PSEUDO-OPERAZIONI DI GOVERNO INTERNO

Esistono varie pseudo-operazioni di governo interno, che influenzano il funzionamento dell'assemblatore ed il listing del suo programma piuttosto che l'uscita del programma stesso. Comuni pseudo-operazioni di governo comprendono:

1) END, che indica la fine del programma sorgente in programma assembly.
2) LIST, che dice all'assemblatore di stampare il programma sorgente. Alcuni assemblatori permettono variazioni quali NO LIST o LIST SYMBOL TABLE per evitare listing lunghi e ripetitivi.
3) NAME e TITLE, che stampano un nome nella parte alta di ogni pagina del listing.
4) PAGE e SPACE, che fanno saltare alla prossima pagina o alla prossima linea, rispettivamente, e migliorano l'aspetto del listing, rendendolo più facile da leggere.
5) PUNCH, che trasferisce il codice oggetto successivo al perforatore di nastro di carta. Questa pseudo-operazione può in qualche caso essere una scelta sbagliata e quindi non necessaria.

LABEL CON PSEUDO-OPERAZIONI

Spesso gli utilizzatori si domandano se e quando assegnare una label ad una pseudo-operazione. Queste sono le nostre raccomandazioni:

1) Tutte le pseudo-operazioni EQUATE devono avere label, altrimenti non avrebbero alcun senso, poichè il loro scopo è di definire il significato delle label.
2) Le pseudo-operazioni DATA e RESERVE di solito hanno label. La label identifica la prima locazione di memoria usata o assegnata.
3) Le altre pseudo-operazioni potrebbero non avere label. Alcuni assemblatori permettono alle altre pseudo-operazioni di avere label, ma il significato della label varia. Raccomandiamo di evitare questa norma.

INDIRIZZI E CAMPO OPERANDO

Gran parte degli assemblatori permettono al programmatore molta libertà nel descrivere il contenuto del campo di indirizzo di Operando. Ma ricordate che l'assemblatore ha nomi di registri e di istruzioni incorporati e può averne altri.

Alcune opzioni comuni per il campo operando sono:

DATI O INDIRIZZI DECIMALI

1) Numeri decimali

Gran parte degli assemblatori presuppongono che tutti i numeri siano decimali a meno che non diversamente indicato. In tale modo:

```
ADD     100
```

significa «somma il contenuto della locazione di memoria 100 in decimale al contenuto dell'Accumulatore».

2) Altri sistemi numerici

ALTRI SISTEMI NUMERICI

Gran parte degli assemblatori accetteranno anche ingressi binari, ottali o esadecimali. Ma dovete identificare questi sistemi numerici in qualche maniera, per es. facendo precedere o seguire il numero con un carattere o una lettera di identificazione:

B o % per il binario
O, Q, C o @ per l'ottale (evitiamo l'O perchè confondibile con lo zero)
H o $ per l'esadecimale
D per il decimale. D può essere tralasciato; è il caso dell'omissione.

Gli assemblatori richiedono generalmente numeri esadecimali inizianti con digit decimale (per es. 0A36 invece di A36) per distinguere tra numeri, nomi e label. È buona norma caricare numeri nella base con significato più chiaro — cioè costanti decimali in decimale; indirizzi e numeri BCD in esadecimale, configurazioni di mascheratura o uscita di bit in binario se sono corte e in esadecimale se sono lunghe.

3) Nomi simbolici

Nel campo operando possono comparire dei nomi; questi saranno trattati come i dati che essi rappresentano. Ma ricordate che **c'è differenza tra dati ed indirizzi.** La sequenza:

```
FIVE    EQU    5
        ADD    FIVE
```

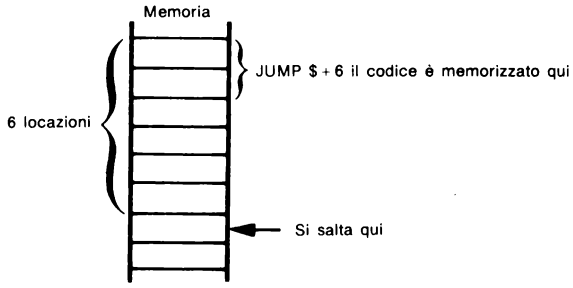
sommerà il contenuto della locazione di memoria 5 (non necessariamente il numero 5) al contenuto dell'Accumulatore.

4) Valore corrente del contatore di locazione (riportato di solito con un * o con $).

Questo è principalmente utile nelle istruzioni Jump; per esempio:

```
JUMP    $ + 6
```

provoca un salto alla locazione di memoria che è sei parole oltre la parola contenente il primo byte dell'istruzione JUMP:



Gran parte dei microprocessori hanno molte istruzioni e due o tre parole. In tale modo, vi troverete in difficoltà nel determinare esattamente quanto diversi siano due statement in linguaggio assembly. Perciò, usando scostamenti dal Contatore di Locazione si hanno come risultato degli errori che si possono evitare usando label.

5) Codici di carattere

CARATTERI ASCII

Gran parte degli assemblatori permettono di caricare testi in stringhe ASCII. Tali stringhe possono essere delimitate sia con semplici che con doppie virgolette; inoltre le stringhe usano un simbolo d'inizio o di fine come A o C. Alcuni assemblatori permettono pure stringhe EBCDIC.

Vi raccomandiamo di usare stringhe di caratteri lungo tutto il testo. Ciò migliora la chiarezza e la leggibilità del programma.

6) Combinazione da 1) a 5) con operatori aritmetici, logici o speciali.

ESPRESSIONI ARITMETICHE E LOGICHE

Quasi tutti gli assemblatori permettono semplici combinazioni aritmetiche, quale START + 1. Alcuni assemblatori permettono pure la moltiplicazione, la divisione, funzioni logiche, scorrimenti, etc. Ci si riferisce a questi come ad espressioni. Si noti che l'assemblatore valuta le espressioni al momento dell'assemblaggio. Anche se un'espressione nel campo operando può comprendere la moltiplicazione, non si può usare la moltiplicazione logica del vostro programma — a meno che non scriviate un sottoprogramma per quello scopo specifico.

Gli assemblatori si differenziano a seconda di quali espressioni accettino e di come le interpretano. Espressioni complesse rendono un programma difficile da leggere e da capire.

Abbiamo fatto alcune raccomandazioni nel corso di questo capitolo ma le ripeteremo e ne aggiungeremo altre qui. In generale, **l'utilizzatore dovrebbe dare importanza alla chiarezza e alla semplicità.** Non c'è alcun pedaggio per essere un esperto nelle implicazioni degli assemblatori o nell'avere l'espressione più complessa nel blocco. **Suggeriamo il seguente approccio:**

1) Usare il più chiaro codice di carattere e sistema numerico per i dati. Maschere e numeri BCD decimali, caratteri ASCII in ottale e ordinarie costanti numeriche in esadecimale non servono ad alcun scopo e perciò non si devono usare.
2) Ricordatevi di distinguere dati da indirizzi.
3) Non usare scostamenti dal Contatore di Locazione.
4) Tenere le espressioni semplici e ovvie. Non appoggiatevi a oscure caratteristiche dell'assemblatore.

ASSEMBLAGGIO CONDIZIONATO

Alcuni assemblatori vi permettono di inserire o di escludere parti del programma sorgente, in funzione di condizioni esistenti all'istante dell'assemblaggio. Questo è denominato assemblaggio condizionato; dà all'assemblatore un po' della flessibilità di un compilatore. **Gran parte degli assemblatori per microcomputer hanno limitate possibilità di assemblaggio condizionato. Una forma usuale è:**

```
IF COND
.
PROGRAMMA CONDIZIONATO
.
.
ENDIF
```

Se l'espressione COND è vera all'istante dell'assemblaggio, nel programma vengono inserite le istruzioni comprese tra IF e ENDIF (due pseudo-operazioni).

Tipici impieghi di assemblaggio condizionato sono:

1) Inserire o escludere variabili supplementari.
2) Mettere della diagnostica o condizioni speciali nei cicli di test.
3) Permettere dati di varie lunghezze di bit.
4) Creare versioni specializzate di un programma comune.

Purtroppo, l'assemblaggio condizionato tende ad appesantire i programmi e a renderli difficile da leggere. Utilizzate l'assemblaggio condizionato solamente nei casi strettamente necessari.

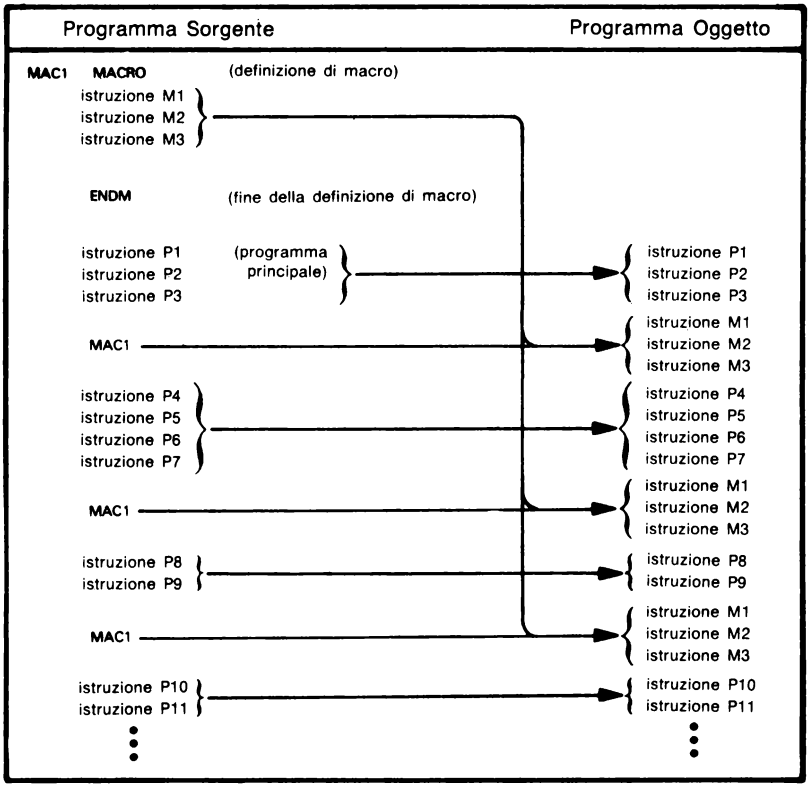
MACRO

DEFINIZIONE DI UNA SEQUENZA DI ISTRUZIONI

Sovente scoprirete che delle particolari sequenze di istruzioni si ripetono diverse volte in un programma sorgente. Sequenze di istruzioni ripetute possono riflettere la necessità della vostra logica di programma, oppure possono costituire una compensazione della carenza nell'insieme d'istruzioni del vostro microprocessore. Si può evitare di trascrivere ripetutamente la medesima sequenza di istruzioni mediante l'impiego di una macro.

Le macro vi consentono di assegnare un nome ad una sequenza di istruzioni. Utilizzate poi il nome della macro nel vostro programma sorgente invece della sequenza di istruzioni ripetuta.

L'assemblatore sostituirà il nome della macro con l'appropriata sequenza di istruzioni. Questo può essere rappresentato nel modo seguente:



Le macro non sono identiche ai sottoprogrammi. Un sottoprogramma è presente una volta sola in un programma, ed è l'esecuzione del programma stesso che porta al sottoprogramma. Una macro viene espansa in una vera e propria sequenza di istruzioni ogni volta che si presenta la stessa macro; pertanto il macro non provoca alcun salto di programma.

Le macro hanno i seguenti vantaggi:

VANTAGGI DELLE MACRO

1) Programmi sorgente più brevi.
2) Documentazione del programma migliore.
3) Utilizzo di sequenze di istruzioni già provate – una volta che la macro è stata già provata, si è sicuri di ottenere una sequenza di istruzioni priva di errori ogni volta che si impiega la macro.
4) Modifiche più semplici. Modificate la definizione della macro e l'assemblatore vi esegue la variazione ogni volta che si usa la macro.
5) Inserimento di comandi, parole chiave, o altre istruzioni dal calcolatore nell'insieme delle istruzioni fondamentali. Si utilizza la macro come estensione delle istruzioni.

Gli svantaggi delle macro sono:

SVANTAGGI DELLE MACRO

1) Ripetizione delle medesime sequenze di istruzioni dato che la macro viene espansa ogni volta che viene utilizzata.
2) Una singola macro può produrre una grande quantità di istruzioni.
3) Mancanza di standardizzazione che può rendere il programma difficile da leggere e da comprendere.
4) Effetti possibili sui registri e sui flag che non possono essere chiaramente stabiliti.

VARIABILI LOGICHE O GLOBALI

Un problema che si presenta consiste nel fatto che le variabili utilizzate in una macro vengono rese note solo all'interno di essa (cioè esse sono locali piuttosto che globali). Ciò può spesso creare molta confusione senza nessuna contropartita. È necessario tenere conto di questi problemi nell'uso delle macro.

COMMENTI

Tutti gli assemblatori vi consentono di inserire dei commenti in un programma sorgente. I commenti non hanno alcun effetto sul codice oggetto, ma vi aiutano nella lettura, nella comprensione, e nella documentazione del programma. Un buon commento costituisce una parte essenziale della scrittura di programmi in linguaggio assembly; senza commenti, i programmi sono molto difficili da comprendere.

TECNICHE DI COMMENTO

Tratteremo i metodi di commento insieme alla documentazione in un capitolo successivo, ma riportiamo di seguito alcune direttive:

1) Usate i commenti per spiegare cosa sta facendo il programma e non cosa fanno le istruzioni.
   I commenti dovrebbero dire cose quali «LA TEMPERATURA HA SUPERATO IL LIMITE?», «LINE FEED PER TTY», oppure «ESAMINA L'INTERRUTTORE DI CARICAMENTO».
   I comandi non dovrebbero dire cose quali «AGGIUNGI 1 ALL'ACCUMULATORE», «SALTA A START», oppure «GUARDA IL CARRY». Dovreste descrivere in che modo il programma sta influenzando il sistema; gli effetti interni sulla CPU sono raramente di particolare interesse.
2) Tenete i commenti brevi e pertinenti. I dettagli dovrebbero essere disponibili altrove nella documentazione.
3) Commentate tutti i punti chiave.
4) Non commentate le istruzioni o le sequenze standard che modificano i contatori e i pun-

tatori; fate particolare attenzione alle istruzioni che possono non avere un significato ovvio.

5) Non usate abbreviazioni poco chiare.
6) Rendete i commenti nitidi e leggibili.
7) Commentate tutte le definizioni, descrivendo i loro scopi. Inoltre indicate tutte le tabelle e le aree di immagazzinamento dei dati.
8) Commentate i pezzi del programma come pure le istruzioni particolari.
9) Siate coerenti nella vostra terminologia. Potete (dovreste) essere ripetitivi; non avete la necessità di consultare un dizionario di sinonimi.
10) Scrivete delle note per voi stessi nei punti che trovate confusi, ad es. «RICORDA CHE IL CARRY È STATO POSIZIONATO A 1 DALL'ULTIMA ISTRUZIONE». Potreste stendere questa nota nella documentazione finale.

Con un programma ben commentato è semplice lavorare. Recupererete il tempo speso per la fase di commento molte più volte. Cercheremo di rappresentare un buon stile di commento negli esempi di programmazioni, sebbene si esageri nel commento stesso a scopo istruttivo.

TIPI DI ASSEMBLATORI

Sebbene tutti gli assemblatori eseguano gli stessi compiti, le loro implementazioni variano grandemente. Non tenteremo di descrivere tutti i tipi di assemblatori esistenti; definiremo semplicemente i termini ed indicheremo alcune alternative.

CROSS-ASSEMBLATORE

Un cross-assemblatore è un assemblatore che funziona su un computer diverso da quello per il quale esso assembla i programmi oggetto.

Il computer sul quale il cross-assemblatore funziona è tipicamente un grosso computer con un vasto supporto software e con periferiche veloci — come un IBM 360 o 370, un Univac 1108, o un Burroughs 6700. Il computer per il quale il cross-assemblatore assembla i programmi è tipicamente un microcomputer come lo Z80 o il MC6800. La maggior parte dei cross-assemblatori vengono scritti in FORTRAN in modo da renderli portabili.

ASSEMBLATORE RESIDENTE

Un auto-assemblatore o assemblatore residente è un assemblatore che funziona sul computer per il quale esso assembla i programmi. L'auto-assemblatore richiederà della memoria e delle periferiche, e può funzionare alquanto lentamente.

MACRO-ASSEMBLATORE

Un macro-assemblatore è un assemblatore che permette di definire sequenze di istruzioni come macro.

MICRO-ASSEMBLATORE

Un micro-assemblatore è un assemblatore utilizzato per scrivere i microprogrammi che definiscono l'insieme delle istruzioni di un computer. La microprogrammazione non ha niente a che fare specificatamente con i microcomputer.

META-ASSEMBLATORE

Un meta-assemblatore è un assemblatore che può trattare molti insiemi diversi di istruzioni. L'utilizzatore deve definire il particolare insieme di istruzioni da utilizzare.

ASSEMBLATORE AD UN PASSAGGIO

Un assemblatore ad un passaggio è un assemblatore che passa attraverso il programma in linguaggio assembly solamente una volta. Un tale assemblatore deve riuscire in qualche modo a determinare i riferimenti diretti, ad es. le istruzioni di Salto che utilizzano delle label che appaiono più avanti nel programma sorgente, cioè che non sono state ancora definite.

ASSEMBLATORE A DUE PASSAGGI

Un assemblatore a due passaggi è un assemblatore che passa due volte attraverso il programma sorgente in linguaggio assembly. La prima volta l'assemblatore raccoglie e definisce semplicemente tutti i simboli; la seconda volta esso sostituisce i riferimenti con le definizioni reali. Un assemblatore a due passaggi risolve la maggior parte dei problemi relativi al riferimento diretto. Tuttavia, l'espansione del macro e l'assemblaggio condizionato possono causare dei problemi. Su di alcune grosse macchine sono necessari sette o più passaggi per garantire che tutti i riferimenti diretti siano determinabili. Un assemblatore a due passaggi può risultare piuttosto lento se non è disponibile nessun immagazzinamento di riserva (come un floppy disk); quindi l'assemblatore deve fisicamente leggere il programma due volte da un elemento d'ingresso lento (come un lettore di nastro di carta per telescrivente). La maggior parte degli assemblatori basati sui microprocessori necessitano di due passaggi.

ERRORI

Gli assemblatori forniscono normalmente messaggi di errore, spesso consistenti in una singola lettera codificata. Alcuni errori tipici sono:

1) Nome indefinito (spesso un errore ortografico o una definizione omessa).
2) Carattere non ammesso (ad es. un 2 in un numero binario).
3) Formato non ammesso (delimitatore errato od operandi sbagliati).
4) Espressione non valida (ad es. due operatori in una riga).
5) Valore non ammesso (solitamente troppo grande).
6) Mancanza dell'operando.
7) Definizione doppia (cioè due valori diversi assegnati ad un nome).
8) Label non ammessa (ad es. una label su una pseudo-operazione che non può possederne una).
9) Mancanza della label.
10) Codice operativo indefinito.

Nell'interpretazione degli errori nell'assemblatore, dovete ricordare che l'assemblatore può procedere per una strada sbagliata se trova una lettera isolata, uno spazio in più, o una punteggiatura scorretta. Molti assemblatori proseguiranno allora ad interpretare male le successive istruzioni e a produrre messaggi d'errore senza significato. Osservate sempre il primo errore molto attentamente; quelli successivi possono dipendere da esso. Una certa accortezza e una coerente aderenza ai formati standard elimineranno molti errori fastidiosi.

CARICATORI

Il caricatore è il programma che preleva realmente l'uscita (codice oggetto) dall'assemblatore e la colloca nella memoria. I caricatori variano dal tipo molto semplice al più complesso. Ne descriveremo alcuni tipi differenti.

CARICATORE BOOTSTRAP

Un caricatore del tipo bootstrap è un programma che impiega un po' delle sue prime istruzioni per caricare la restante parte di se stesso o un altro programma caricatore nella memoria. Il caricatore di tipo bootstrap può essere presente in ROM, o potete aver la necessità di farlo caricare nella memoria del computer mediante l'uso di interruttori del pannello frontale. L'assemblatore può collocare questo tipo di caricatore all'inizio del programma oggetto che esso produce.

CARICATORE DI RIALLOCAZIONE

Un caricatore di rilocazione può caricare i programmi in qualsiasi punto della memoria. Esso carica tipicamente ciascun programma nello spazio di memoria immediatamente seguente a quello utilizzato dal precedente programma. Tuttavia i programmi stessi debbono essere in grado di essere

spostati qua e là in questa maniera, vale a dire essi debbono essere riallocabili. Un caricatore assoluto, al contrario, collocherà sempre i programmi nella medesima area di memoria.

CARICATORI DI LINKING

Un caricatore di linking carica i programmi e i sottoprogrammi che sono stati separatamente assemblati; esso determina i riferimenti esterni – cioè, una istruzione in un modulo che si riferisca ad una label con un altro modulo. I programmi oggetto caricati da un caricatore linking debbono essere prodotti da un assemblatore che permetta e marchi i riferimenti esterni.

LINK EDITOR

Un metodo alternativo è quello di separare le funzioni di linking e di caricamento delle funzioni e di ottenere il linking realizzato da un programma chiamato link editor.

BIBLIOGRAFIA

Una monografia completa sui macro è M. Campbell Welly, An Introduction to Macros, American Elsevier, New York, 1973.

La microprogrammazione è concettualmente descritta in An Introduction to Microcomputers: Volume 1 – Basic Concepts, Capitolo 4. Una descrizione più tecnica si trova in A.K. Agrawala e T.G. Rauscher, Foundations of Microprogramming, Academic Press, New York, 1976.

Potete trovare descrizioni più dettagliate su assemblatori e caricatori in D.W. Barron, «Assemblers and Loaders», American Elsevier, New York, 1972 e in C.W. Gear, Computer Organization and Programming Mc Graw-Hill, New York, 1974.

Capitolo 3

L'INSIEME DI ISTRUZIONI DEL LINGUAGGIO ASSEMBLY DELLO Z80

Ora siamo pronti per iniziare a scrivere programmi in linguaggio assembly. In questo capitolo cominceremo col definire le singole istruzioni dell'insieme (set) di istruzioni in linguaggio assembly dello Z80, più le regole di sintassi dell'assemblatore Zilog.

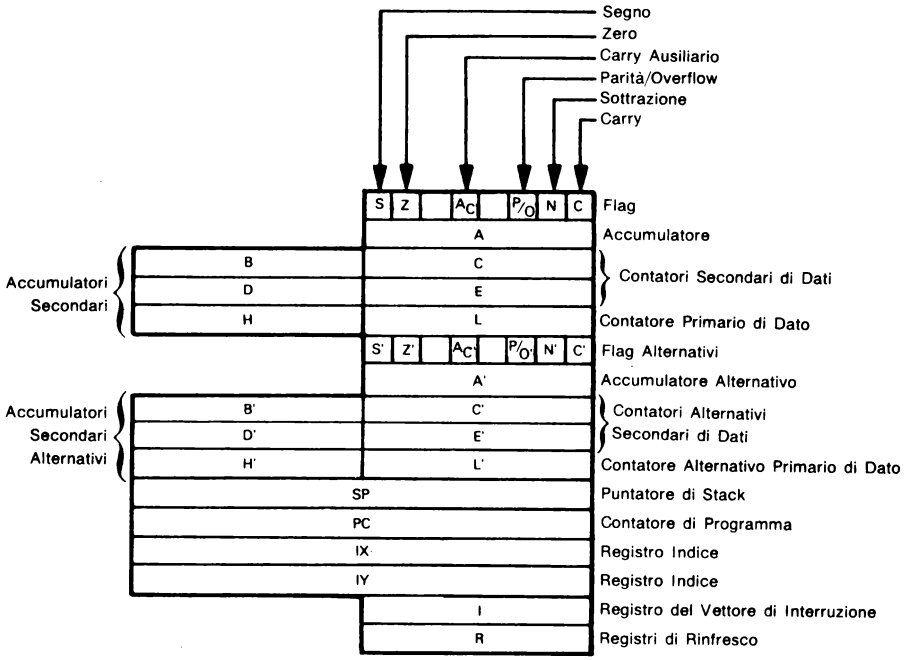
In questo libro non tratteremo alcun aspetto dell'hardware del microcomputer, dei segnali, delle interfacce o dell'architettura della CPU. Queste informazioni sono descritte in dettaglio in An Introduction to Microcomputers: Volume 2 – Some Real Microprocessor e Volume 3 – Some Real Support Devices, mentre Z80 Programmazione per il progetto logico tratta del linguaggio assembly come estensione di logica digitale. In questo libro **guarderemo le tecniche di programmazione dal punto di vista del programmatore in linguaggio assembly, per il quale i piedini e i segnali sono non rilevanti e non esistono importanti differenze tra un minicomputer ed un microcomputer.**

Nei successivi capitoli di questo libro, si descriveranno gli interrupt, l'accesso diretto in memoria e l'architettura dello Stack dello Z80, insieme con discussioni sulla programmazione in linguaggio assembly degli stessi soggetti. Questo capitolo contiene una definizione dettagliata di ogni istruzione in linguaggio assembly. Queste definizioni sono identiche a quelle trovate nel Capitolo 6 di Z80 Programmazione per il progetto logico.

La descrizione dettagliata delle singole istruzioni è preceduta da una trattazione generale dell'insieme di istruzioni dello Z80 che divide le istruzioni in quelle comunemente usate, in quelle frequentemente usate e in quelle raramente usate. Se siete dei programmatori con esperienza di linguaggio assembly, questa classificazione non è particolarmente importante – e, in funzione dei vostri pregiudizi sulla programmazione, può anche essere imprecisa. Se siete dei programmatori nuovi al linguaggio assembly, vi raccomandiamo di cominciare a scrivere programmi usando solo istruzioni della categoria «comunemente usate». Una volta che avete assimilato i concetti della programmazione in linguaggio assembly, potete esaminare le altre istruzioni e usarle appropriatamente.

REGISTRI DELLA CPU E FLAG DI STATO

I registri della CPU e i flag di stato dello Z80 possono essere illustrati come segue:



L'Accumulatore è la destinazione e la sorgente principale per istruzioni ad uno e due operandi. Per esempio, i trasferimenti di dati più corti e più veloci tra CPU e dispositivi di I/O sono realizzati mediante l'Accumulatore. Inoltre, un numero maggiore di istruzioni di Riferimento alla Memoria spostano dati tra Accumulatore e memoria piuttosto che tra un altro registro e memoria. Tutte le istruzioni aritmetiche e Booleane ad 8 bit prendono un solo operando dell'Accumulatore e riportano il risultato nell'Accumulatore. Un'istruzione perciò deve **caricare l'Accumulatore prima che lo Z80 possa effettuare una operazione aritmetica o Booleana ad 8 bit.**

I registri B, C, D, E, H e L sono tutti registri secondari. Si può accedere a dati memorizzati in uno qualunque di questi sei registri con eguale facilità; tali dati possono essere spostati verso qualunque altro registro o usati come secondo operando in istruzioni a due operandi.

Esistono, tuttavia, alcune importanti differenze nelle funzioni dei Registri B, C, D, E, H e L.

I Registri H e L sono il principale Puntatore di Dato per lo Z80. Cioè, normalmente userete questi due registri per conservare l'indirizzo di memoria a 16 bit del dato da indirizzare. Il dato può essere trasferito tra un qualunque registro e la locazione di memoria indirizzata da H e L. Poichè HL è il Puntatore di Dato primario, spesso richiede meno byte di codice oggetto e meno cicli di istruzione per compiere operazioni mediante se stesso. Il programmatore dello Z80 dovrebbe provare ad indirizzare memorie dati mediante i registri H e L ogniqualvolta è possibile.

Nella logica del vostro programma, riservate sempre i Registri H e L per contenere un indirizzo di memoria dati.

I Registri B, C, D ed E forniscono un immagazzinamento di dati secondario; frequentemente, in uno di questi quattro registri si immagazzina il secondo operando di istruzioni a due operandi. (Il primo operando è immagazzinato nell'Accumulatore, che è pure la destinazione del risultato).

Esiste **un numero limitato di istruzioni** che **trattano i Registri B e C, o D ed E, come Puntatore di Dato a 16 bit.** Ma queste istruzioni spostano dati solo tra memoria e l'Accumulatore.

Nella vostra logica di programma dovreste normalmente usare i Registri B, C, D ed E come immagazzinamento temporaneo di dati o indirizzi.

I Registri IX e IY sono registri indice. Essi forniscono una limitata capacità di indicizzamento del tipo descritto in An Introduction to Microcomputers: Volume 1 per brevi istruzioni.

I registri alternativi F', A', B', C', D', E', H' e L' forniscono un insieme duplicato di registri di uso generale. Solo due istruzioni Exchange a byte singolo selezionano e de-selezionano tutti i registri alternativi; un'istruzione scambia AF con l'alternativa AF' come coppia di registri, ed un'istruzione scambia BC, DE ed HL con le alternative BC', DE' ed HL'. Una volta effettuata la scelta, tutte le operazioni successive sui registri sono effettuate sull'insieme attivo fino a che il prossimo scambio non scelga l'insieme non attivo. **I registri alternativi possono essere riservati per usi dove si richiede una veloce risposta all'interrupt.** Oppure essi possono essere usati nel modo desiderato dal programmatore.

C'è un numero di istruzioni che gestisce dati a 16 bit contemporaneamente. Queste istruzioni fanno riferimento a coppie di registri della CPU come segue:

F	e	A
B	e	C
D	e	E
H	e	L
F'	e	A'
B'	e	C'
D'	e	E'
H'	e	L'
Byte di ordine alto		Byte di ordine basso

La combinazione dell'Accumulatore e dei flag, vista come unità a 16 bit, è usata solo per operazioni di Stack e per commutazioni dei registri alternativi. Operazioni aritmetiche accedono a B e a C, a D e ad E, oppure ad H ed L come unità di dato a 16 bit.

Il flag di stato Carry riporta il bit più significativo in una qualunque operazione aritmetica. Il flag Carry è pure interessato nelle istruzioni di Shift; è azzerato da istruzioni Booleane.

Il flag Substract è stato progettato per uso interno durante operazioni di regolazione decimale. Questo flag è posizionato ad 1 per tutte le istruzioni Substract e posizionato a 0 per tutte le istruzioni Add.

Il flag Parity/Overflow è un flag di uso molteplice, in funzione dell'operazione che si sta compiendo. Per operazioni aritmetiche, esso è un flag di overflow. Per operazioni di ingresso, di rotazione e Booleane, è un flag di parità, con 1 = parità pari e 0 = parità dispari. Durante operazioni di trasferimento e di ricerca di blocco, rimane posizionato ad 1 finché il contatore dei byte non è decrementato a zero; quindi è posizionato a zero. È pure posizionato allo stato corrente del flip-flop (IFF2) di abilitazione delle istruzioni quando si esegue una istruzione LD A,I oppure LD A,R.

Il flag Zero è posizionato ad 1 quando una operazione aritmetica o Booleana genera un risultato zero. Lo stato Zero è posizionato a 0 quando tali operazioni generano un risultato diverso da zero.

Il flag di stato Sign acquisisce il valore del bit più significativo del risultato conseguente l'esecuzione di un'istruzione aritmetica o Booleana.

Il flag di stato Auxiliary Carry mantiene ogni riporto dal bit 3 al bit 4 risultante dall'esecuzione di un'istruzione aritmetica. Lo scopo di questo flag di stato è di semplificare le operazioni Binary-Coded-Decimal (BCD); questo è l'uso tipico di un flag di stato Auxiliary Carry come descritto in An Introduction to Microcomputers: Volume 1, Capitolo 3.

Tutti i flag di stato precedenti mantengono il loro valore corrente fino a che non venga eseguita una istruzione che li modifica. Il cambiare opportunamente il valore dell'Accumulatore non fa variare necessariamente il valore dei flag di stato. Per esempio, se il flag Zero è posizionato ad 1 e si esegue un caricamento immediato nell'Accumulatore, cosa che fa sì che l'Accumulatore acquisisca un valore diverso da zero; il valore del flag Zero rimane immutato.

Il Puntatore dello Stack a 16 bit vi permette di effettuare una operazione di Stack, dovunque nella memoria indirizzabile. La dimensione dello Stack è limitata solo dalla quantità di memoria indirizzabile presente. In realtà raramente userete più di 256 byte di memoria per il vostro Stack. Dovreste usare lo Stack per accedere a sottoprogrammi e per elaborare interrupt. Non usare lo Stack per passare parametri a sottoprogrammi. Ciò non è molto efficiente per i limiti dell'insieme di istruzioni dello Z80. Lo Stack dello Z80 è inizializzato al suo indirizzo maggiore. Un Push decrementa il contenuto del Puntatore di Stack; un Pop incrementa il contenuto del Puntatore di Stack.

Il registro del Vettore di Interrupt e il registro di Rinfresco sono registri di scopo specifico non usati normalmente dal programmatore.

Il registro del Vettore di Interrupt è usato per immagazzinare l'indirizzo di pagina di una routine di risposta ad un'interrupt; la locazione della pagina è fornita dal dispositivo interrompente. Questo schema permette di cambiare l'indirizzo della routine di risposta ad un'interrupt fornendo inoltre un tempo di risposta molto veloce per il dispositivo interrompente.

Il registro di Rinfresco contiene un contatore di rinfresco della memoria nei sette bit di peso minore. Questo contatore è automaticamente incrementato dopo ogni fetch d'istruzioni e fornisce il successivo indirizzo di rinfresco per memorie dinamiche. Il bit di ordine maggiore del registro di Rinfresco rimarrà posizionato ad 1 o a 0, in funzione di come è stato caricato con l'ultima istruzione LD R,A.

MODI DI INDIRIZZAMENTO DELLA MEMORIA CON LO Z80

Lo Z80 fornisce ampi modi di indirizzamento. Questi comprendono:

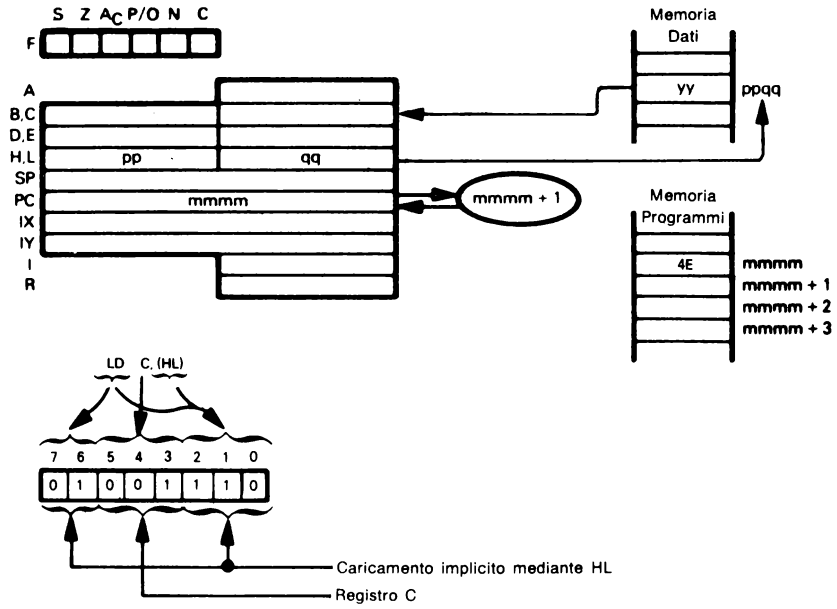
- **Implicito.**
- **Trasferimento di Blocco Implicito con Auto-Incremento/Decremento.**
- **Stack Implicito.**
- **Indicizzato.**
- **Diretto.**
- **Relativo al Programma.**
- **Pagina Base.**
- **Indiretto con Registro.**
- **Immediato.**

Implicito

In un indirizzamento implicito di memoria, i registri A ed L contengono l'indirizzo della locazione di memoria a cui si deve accedere. I dati possono essere spostati tra la locazione di memoria identificata ed ognuno dei sette registri della CPU A, B, C, D, E, H o L. Per esempio l'istruzione

LD C,(HL)

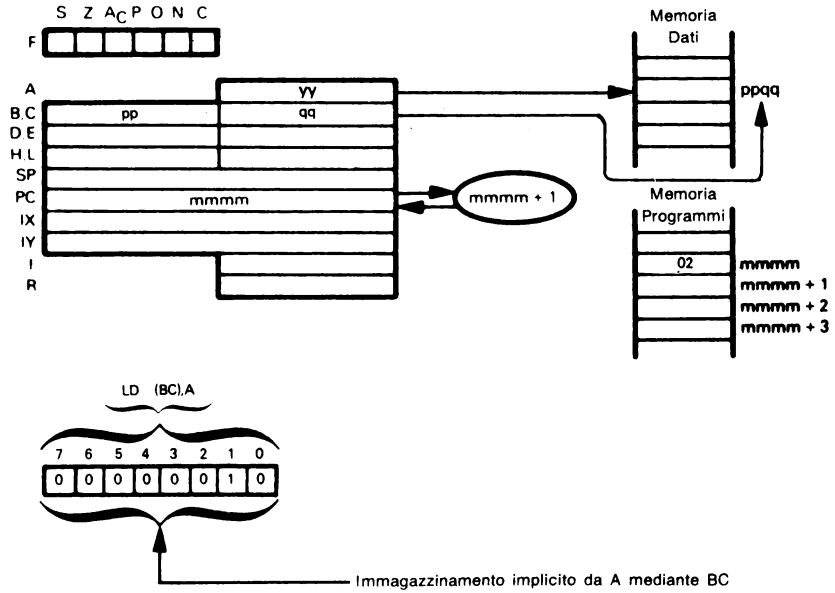
carica il registro C col contenuto della locazione di memoria indirizzata correntemente da HL. Questo si può illustrare come segue:



Un numero limitato di istruzioni usa i Registri B e C o D ed E come Puntatore del Dato. Queste istruzioni spostano i dati tra l'Accumulatore e la locazione di memoria indirizzata dai Registri B e C o dai Registri D ed E. L'istruzione

LD (BC),A

immagazzina il contenuto di A nella locazione di memoria indirizzata dalla Coppia di Registri BC. Questo è illustrato come segue:



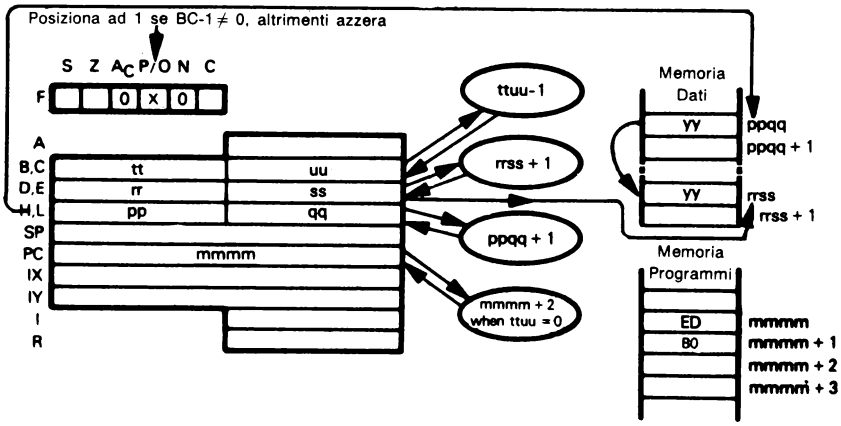
Trasferimento di Blocco Implicito con Auto-Incremento/Decremento

Le istruzioni di Ricerca e di Trasferimento di Blocco agiscono su un blocco di dati la cui dimensione è posizionata dal programmatore come contenuto della coppia di registri BC. In questa forma di indirizzamento, un dato di un byte viene spostato dalla locazione di memoria indirizzata da HL nella locazione di memoria indirizzata da DE; poi si incrementano HL e DE e si decrementa BC. Il trasferimento dei dati continua finché BC non arriva a zero, dopodiché l'istruzione finisce. Variazioni permettono che altre istruzioni seguino ogni trasferimento di dati, con il programmatore che fornisca il ritorno al loop; auto-decremento di HL e di DE invece di auto-incremento, ed un insieme complementare di istruzioni di Ricerca di Blocco che confrontano il byte di memoria indirizzato da HL con il contenuto del registro A, posizionando ad 1 un flag se si trova una corrispondenza.

L'istruzione Load, Increment e Repeat

LDIR

è illustrata come segue:



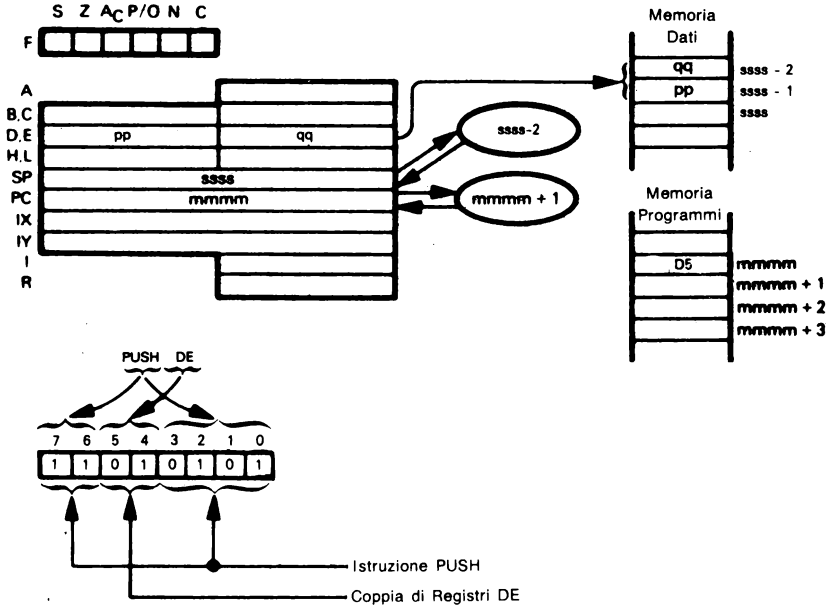
Si fornisce un gruppo similare di istruzioni di Ingresso/Uscita, permettendo che un blocco di dati sia fatto entrare o uscire tra la memoria di un dispositivo di I/O. Il numero della porta di I/O è preso dal contenuto del registro C, con il solo registro B usato come contatore di byte. La memoria è indirizzata da HL.

Stack Implicito

Poiché lo Stack è parte della memoria di Lettura-Scrittura, dobbiamo considerare le istruzioni di Stack come istruzioni di Riferimento alla Memoria. **Le istruzioni Push e Pop spostano due byte di dati tra una coppia di registri e la locazione di memoria indirizzata dal Puntatore di Stack,** cioè la sommità corrente dello Stack. L'indirizzo di Stack dello Z80 è decrementato da ogni Push e incrementato da ogni Pop. L'istruzione

PUSH DE

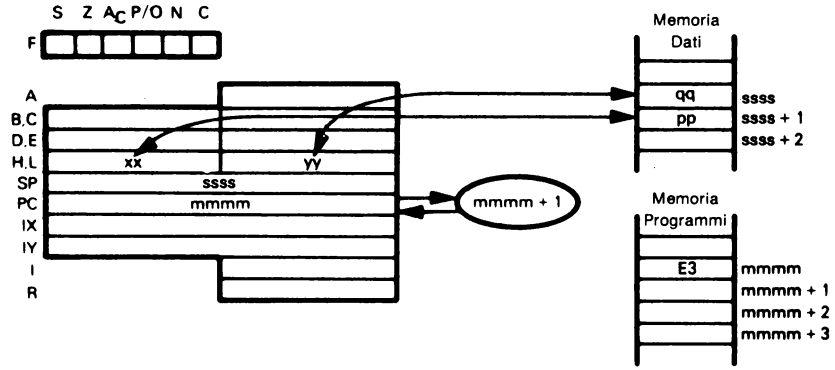
è illustrata come segue:



Lo Z80 ha pure istruzioni che scambiano i due byte superiori dello stack con un registro a 16 bit – HL o uno dei due registri indice. L'istruzione

EX (SP), HL

è illustrata come segue:

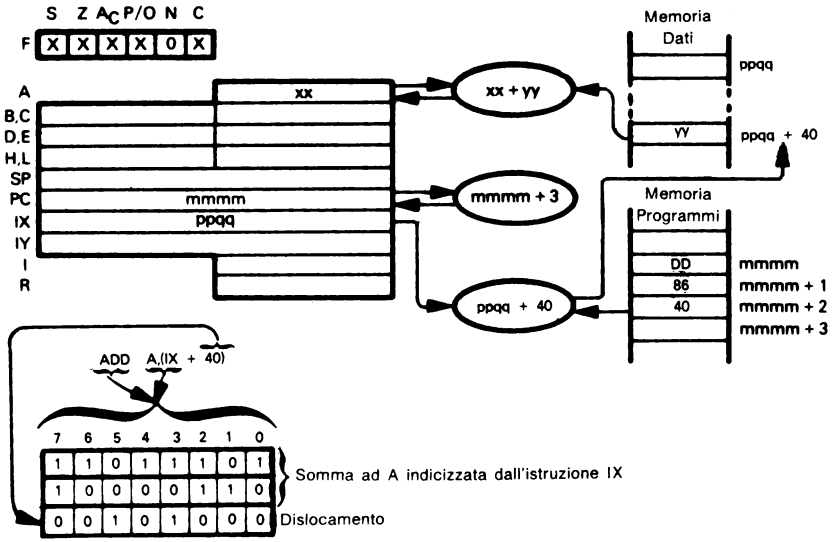


Indicizzato

Lo Z80 ha due registri indice a 16 bit, chiamati IX e IY. Possono essere usati intercambiabilmente. Tutte le operazioni di riferimento alla memoria per le quali si può specificare (HL) possono essere specificate in alternativa come operazioni indicizzate. La differenza tra indirizzamento implicito utilizzante HL e indirizzamento indicizzato utilizzante IX e IY consiste nel fatto che **l'operando di indice comprende un valore di spostamento che viene sommato all'indirizzo indice.** Nell'istruzione

ADD A,(IX + 40H)

l'indirizzo di memoria è la somma del contenuto dei registri IX e 40_{16}. Questo può essere illustrato come segue:



Diretto

Si può usare un indirizzamento diretto per caricare l'Accumulatore con un valore ad 8 bit della memoria, per caricare BC, DE, HL, SP, IX o IY con un valore di memoria a 16 bit, e saltare a o chiamare sottoprogrammi direttamente in una locazione di memoria. L'indirizzo diretto a 16 bit è immagazzinato negli ultimi due byte dell'istruzione, nell'ordine byte basso, byte alto (ciò è l'opposto dello schema standard alto-basso).

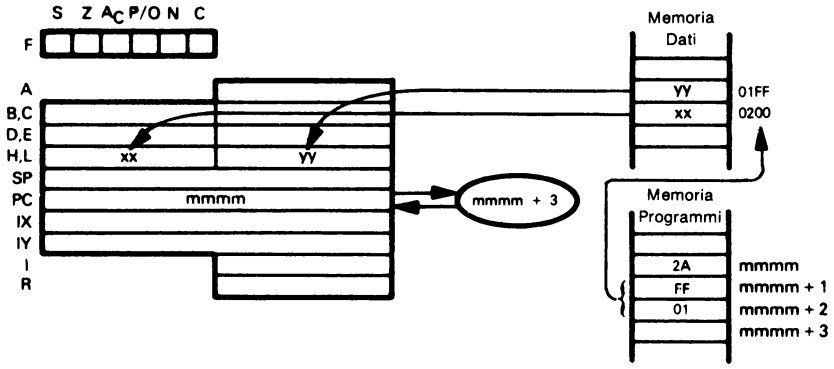
L'istruzione

LD A,(NEXT)

carica il registro A col contenuto della locazione di memoria indirizzata dalla label NEXT. L'istruzione

LD HL,(1FFH)

carica nel registro L il contenuto della locazione di memoria $01FF_{16}$ e nel registro H il contenuto della locazione di memoria 0200_{16}. Questo può essere illustrato come segue:



LD HL,(1FFH)

7	6	5	4	3	2	1	0	
0	0	1	0	1	0	1	0	Istruzione di Caricamento Diretto di HL
1	1	1	1	1	1	1	1	Indirizzo diretto - byte minore
0	0	0	0	0	0	0	1	Indirizzo diretto - byte maggiore

Le istruzioni di Jump diretto forniscono salti e salti-alle-subroutine, sia incondizionati che condizionati. Queste sono tutte istruzioni a 3 byte, con l'indirizzo diretto immagazzinato nel secondo e nel terzo byte dell'istruzione, come mostrato sopra per Load Direct.

Ci sono altri tre modi di indirizzamento usati dalle istruzioni Branch dello Z80: relativo al programma, pagina base e indiretto con registro. In generale, sono più brevi e/o più veloci dei salti diretti ma possono avere capacità limitate di indirizzamento.

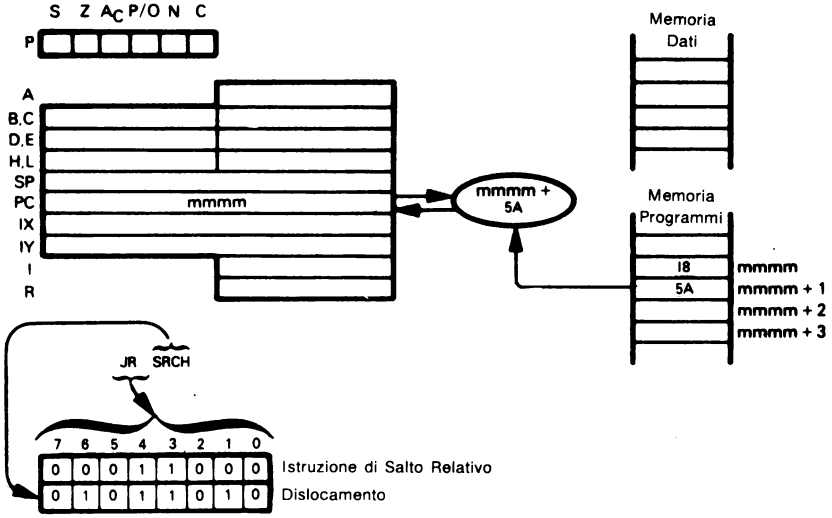
Relativo al Programma

Istruzioni di Jump Relativo forniscono al programma indirizzamento relativo nel campo da −126 a +129 byte a partire dal primo byte dell'istruzione Relativa di Programma. Queste istruzioni sono tutte istruzioni a 2 byte, col valore con segno dello spostamento memorizzato nel secondo byte dell'istruzione. **Esistono salti relativi incondizionati e condizionati, come pure l'istruzione Decrement e Jump If No Zero (DJNZ) che facilita il controllo di loop.**

Data l'istruzione

JR SRCH

supponiamo SRCH sia una label indirizzante una locazione di memoria, posta dopo $5A_{16}$ byte rispetto al byte del codice operativo JR. L'operazione può essere illustrata come segue:



Pagina Base

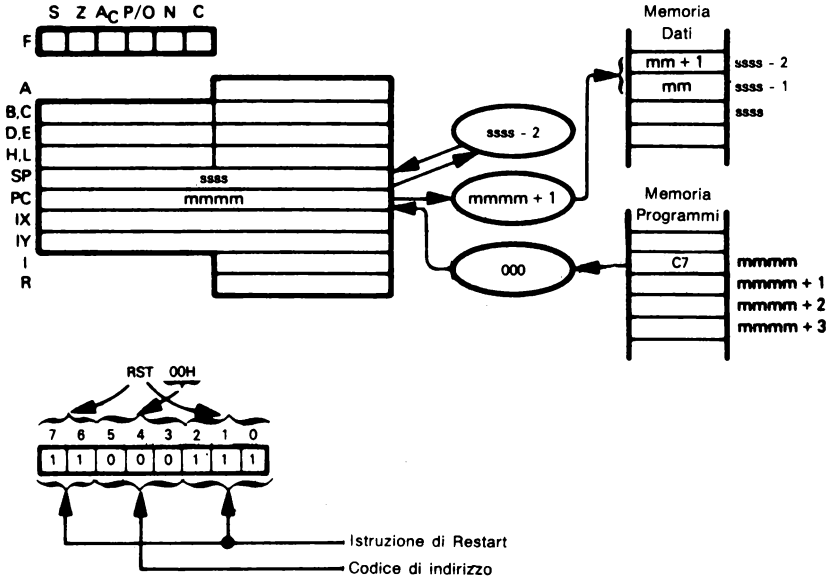
Lo Z80 ha un modo d'**indirizzamento in pagina base modificato** per l'istruzione Restart. Questa è una speciale istruzione Call che **permette ad un'istruzione ad un solo byte di saltare ad uno degli otto sottoprogrammi allocati in più punti specifici nel nucleo di peso minore.** L'indirizzo effettivo è calcolato dal codice a 3 bit memorizzato nell'istruzione, come segue:

Indirizzo Nucleo di Peso Minore	Codice a 3 bit
00H	000
08H	001
10H	010
18H	011
20H	100
28H	101
30H	110
38H	111

Il valore decodificato dell'indirizzo è caricato nel byte di peso minore del Contatore di Programma; il byte di ordine maggiore del Contatore di Programma è posizionato a zero. Per esempio, l'istruzione

RST 00H

è illustrata come segue:



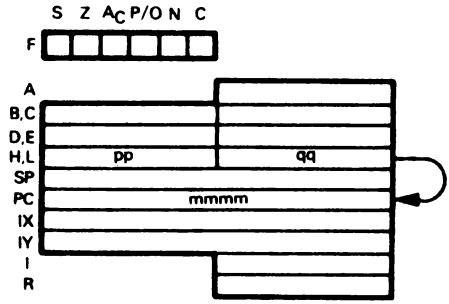
Indiretto con Registro

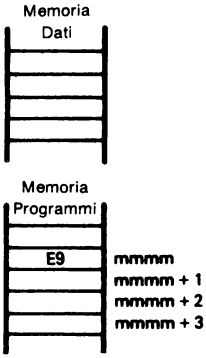
Nell'indirizzamento indiretto standard, una locazione di memoria contiene l'indirizzo effettivo e l'istruzione specifica l'indirizzo della locazione di memoria contenente l'indirizzo effettivo. Nell'indirizzamento indiretto con registro, un registro contiene l'indirizzo effettivo, e l'istruzione specifica quale registro contiene l'indirizzo effettivo. Si noti, per esempio, che per un Load, questo è solo un altro modo di descrivere un indirizzamento implicito. Tuttavia **lo Z80 ha istruzioni Jump che permettono un salto alla locazione di memoria il cui indirizzo è contenuto nel registro specificato.** Questa è una forma di indirizzamento indiretto, ed è descritta separatamente poiché, mentre gran parte dei microcomputer hanno l'indirizzamento indiretto, pochissimi hanno salti indiretti tramite registro.

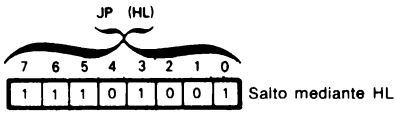
L'istruzione

JP (HL)

stabilisce che si deve fare un salto alla locazione di memoria il cui indirizzo è contenuto in HL. Questo può essere illustrato come segue:







Immediato

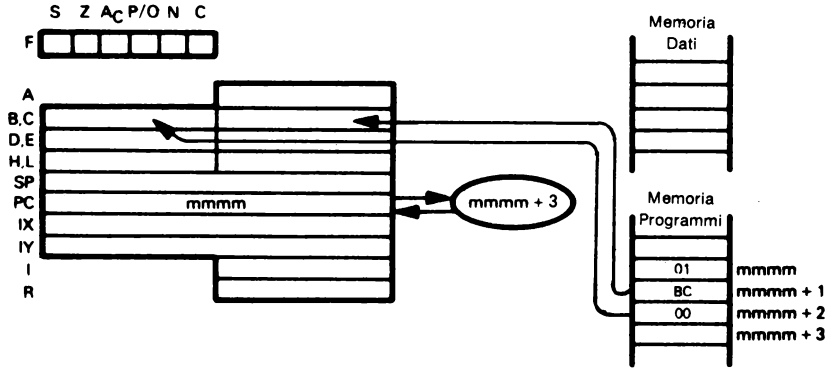
Alcuni testi identificano le istruzioni Immediate come istruzioni di Riferimento alla Memoria. Un'Istruzione Immediata è un'istruzione a 2, 3 o 4 byte nella quale l'ultimo o gli ultimi 2 byte contengono dati fissi che sono caricati in un registro o in una locazione di memoria. **Lo Z80 fornisce istruzioni Immediate per:**

- **caricare un dato ad 8 bit in uno dei registri ad 8 bit.**
- **caricare un dato a 16 bit in una delle coppie di registri o registri a 16 bit.**
- **memorizzare un dato ad 8 bit in una locazione di memoria utilizzando un indirizzamento implicito o indicizzato.**
- **realizzare operazioni aritmetiche e logiche usando l'Accumulatore ed un dato immediato ad 8 bit.**

L'istruzione

LD BC, 0BCH

carica il valore BC_{16} del dato immediato nella Coppia di Registri BC. Questo può essere illustrato come segue:



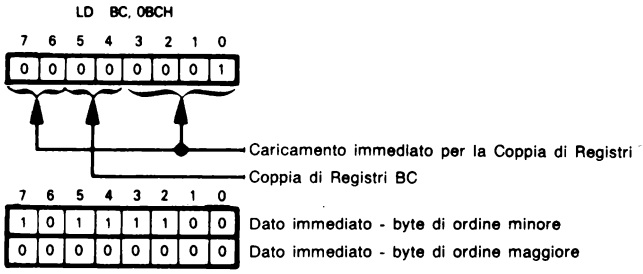


Tabella 3-1. Istruzioni dello Z80 usate frequentemente

Codice di Istruzione		Significato
ADC	A	Add with Carry to Accumulator
ADD		Add
AND		Logical AND
CALL	addr	Call Subroutine
CALL	cond,addr	Call Conditional
CP		Compare
DEC		Decrement
DJNZ		Decrement and Jump If Not Zero
IN		Input
INC		Increment
JR		Jump Relative
JR	cond,addr	Jump Relative Conditional
LD	reg,(HL)	Load Register
LD	A,(addr)	Load Accumulator Direct
LD	data	Load Immediate
LD	(HL),reg	Store Register
LD	(addr),A	Store Accumulator Direct
LD	dst,src	Move Register-to-Register
OUT		Output
POP		Pop from Stack
PUSH		Push to Stack
RET		Return from Subroutine
RET	cond	Return Conditional
RLA		Rotate Accumulator Left Through Carry
RRA		Rotate Accumulator Right Through Carry
SLA		Shift Left Arithmetic
SRL		Shift Right Logical
SUB		Subtract

Tabella 3-2. Istruzioni dello Z80 usate occasionalmente

Codice di Istruzione		Significato
BIT		Test Bit
CPD, CPDR		Compare, Decrement, (Repeat)
CPI, CPIR		Compare, Increment, (Repeat)
CPL		Complement Accumulator
DAA		Decimal Adjust Accumulator
DI		Disable Interrupts
EI		Enable Interrupts
EX		Exchange
HALT		Halt
IND, INDR		Input, Decrement, (Repeat)
INI, INIR		Input, Increment, (Repeat)
JP	addr	Jump
JP	cond,addr	Jump Conditional
LD	A,(BC) or (DE)	Load Accumulator Secondary
LD	HL,(addr)	Load HL Direct
LD	reg,(xy+disp)	Load Register Indexed
LD	rp,(addr)	Load Register Pair Direct
LD	xy,(addr)	Load Index Register Direct
LD	(BC) or (DE),A	Store Accumulator Secondary
LD	(addr),HL	Store HL Direct
LD	(xy+disp),reg	Store Register Indexed
LD	(addr),rp	Store Register Pair Direct
LD	(addr),xy	Store Index Register Direct
LD	(HL),data	Store Immediate to Memory
LD	(xy+disp),data	Store Immediate to Memory Indexed
LDD, LDDR		Load, Decrement, (Repeat)
LDI, LDIR		Load, Increment, (Repeat)
NEG		Negate (Twos Complement) Accumulator
NOP		No Operation
OR		Logical OR
OUTD, OTDR		Output, Decrement, (Repeat)
OUTI, OTIR		Output, Increment, (Repeat)
RES		Reset Bit
RETI		Return from Interrupt
RL		Rotate Left Through Carry
RLC		Rotate Left Circular
RLCA		Rotate Accumulator Left Circular
RR		Rotate Right Through Carry
RRC		Rotate Right Circular
RRCA		Rotate Accumulator Right Circular
SET		Set Bit
SRA		Shift Right Arithmetic
XOR		Logical Exclusive OR

Tabella 3-3. Istruzioni dello Z80 usate raramente

Codice di Istruzione	Significato
ADC HL,rp	Add Register Pair with Carry to HL
CCF	Complement Carry Flag
EXX	Exchange Register Pairs and Alternatives
IM n	Set Interrupt Mode
RETN	Return from Non-Maskable Interrupt
RLD	Rotate Accumulator and Memory Left Decimal
RRD	Rotate Accumulator and Memory Right Decimal
RST	Restart
SBC	Subtract with Carry (Borrow)
SCF	Set Carry Flag
LD A,I	Load Accumulator from Interrupt Vector Register
LD A,R	Load Accumulator from Refresh Register
LD I,A	Store Accumulator to Interrupt Vector Register
LD R,A	Store Accumulator to Refresh Register
LD SP,HL	Move HL to Stack Pointer
LD SP,xy	Move Index Register to Stack Pointer

ABBREVIAZIONI

Ecco le abbreviazioni usate in questo capitolo:

A,F,B,C,D,E,H,L	I registri ad 8 bit. A è l'accumulatore ed F è la Parola dello Stato del Programma.
AF',BC',DE'HL'	Le coppie alternative di registri
addr	Un indirizzo di memoria a 16 bit
x(b)	Bit b di un registro a 8 bit o di una locazione di memoria x
cond	Condizione per un salto di programma. Le condizioni sono: NZ – Non Zero (Z=0) Z – Zero (Z=1) NC – Non carry (C=0) (Nessun Riporto) C – Carry (C=1) (Riporto) PO – Parity Odd (P=0) (Parità dispari) PE – Parity Even (P=1) (Parità pari) P – Sign Positive (S=0) (Segno Positivo) M – Sign Negative (S=1) (Segno Negativo)
data	Un'unità dati binari ad 8 bit
data 16	Un'unità dati binari a 16 bit
disp	Un dislocamento di un indirizzo binario a 8 bit con segno
xx(H)	Gli 8 bit di ordine maggiore di una quantità xx a 16 bit
IV	Registro del vettore d'interruzione (8 bit)
IX,IY	I registri Indice (ciascuno a 16 bit)
xy	L'uno o l'altro dei registri Indice (IX o IY)
LSB	Least Significant Bit (Bit 0) (Bit Meno Significativo)
Label	Un indirizzo a 16 bit di un'istruzione di memoria
xx(LO)	Gli 8 bit di ordine minore di una quantità xx a 16 bit
MSB	Most Significant Bit (Bit 7) (Bit Più Significativo)
PC	Program Counter (Contatore Programma)
port	Un indirizzo di 8 bit di una porta di I/O

pr	Una qualsiasi delle seguenti coppie di registri:		
	BC		
	DE		
	HL		
	AF		
R	Il registro di Refresh (8 bit)		
reg	Uno qualsiasi dei seguenti registri:		
	A		
	B		
	C		
	D		
	E		
	H		
	L		
rp	Una qualsiasi delle seguenti coppie di registri:		
	BC		
	DE		
	HL		
	SP		
SP	Stack Pointer (16 bit) (Puntatore della Catasta)		
xy	L'uno o l'altro dei Registri Indice (IX o IY)		
Codice Oggetto	bbb	Numero in bit da 000 (LSB) a (111) (MSB)	
	ccc	Codice della condizione	000 = non zero
			001 = zero
			010 = nessun riporto
			011 = riporto
			100 = parità dispari
			101 = parità pari
			110 = segno positivo
			111 = segno negativo
	ddd	Registro di destinazione – lo stesso codice di rrr	
	ppqq	Indirizzo di memoria a 16 bit	
	rrr	Registro	111 = A
			000 = B
			001 = C
			010 = D
			011 = E
			100 = H
			101 = L
	sss	Registro sorgente – la stessa codifica di rrr	
	x	Registro indice	0 = IX
			1 = IY
	xx	Coppia di registri	00 = BC
			01 = DE
			10 = HL
			11 = SP (rp) o AF (pr)
	xxx	Codice Restart (000 a 111)	
	yy	Unità di dato binario ad 8 bit	
	yyyy	Unità di dato binario a 16 bit	

Statuses

Lo Z80 ha i seguenti flags di stato:

C – Stato di Carry
Z – Stato di Zero
S – Stato di Segno
P/O – Stato di Parità/Overflow
Ac – Stato di Carry ausiliario
N – Stato di Subtract

Nelle colonne dello stato si usano i seguenti simboli:

X – Il flag è influenzato dall'operazione
(blank) – Il flag non è influenzato dall'operazione
1 – Il flag è posto a 1 dall'operazione
0 – Il flag è posto a 0 dall'operazione
? – Il flag è sconosciuto dopo l'operazione
P – Il flag mostra lo stato di parità
O – Il flag mostra lo stato di overflow
I – Il flag mostra lo stato di interruzione abilitata/disabilitata

[[ ]] Indirizzamento di memoria: 1) il contenuto della locazione di memoria il cui indirizzo è contenuto nel registro designato, 2) una porta di I/O il cui indirizzo è contenuto nel registro designato.

[ ] Il contenuto di un registro o di una locazione di memoria.

Per esempio:

$$[[HL]] \leftarrow [[HL]] + 1$$

indica che il contenuto della locazione di memoria indirizzata dal contenuto di HL è incrementato, mentre

$$[HL] \leftarrow [HL] + 1$$

indica che il contenuto dello stesso registro HL è incrementato.

$\wedge$ AND logico
$\vee$ OR logico
$\forall$ Exclusiv-OR logico
$\leftarrow$ Il dato è trasferito nella direzione della freccia
$\longleftrightarrow$ Il dato è scambiato tra le due locazioni indicate sui due lati della freccia

MNEMONICA DELLE ISTRUZIONI

La Tabella 3-4 dà il sommario dell'insieme di istruzioni dello Z80. La colonna INSTRUCTION mostra il codice mnemonico della istruzione (IN, OUT, LD) e gli operandi, se ci sono, usati col codice mnemonico dell'istruzione.

La parte fissa di un'istruzione in linguaggio assembly è indicata con LETTERE MAIUSCOLE. La parte variabile (dato immediato, numero del dispositivo di I/O, nome del registro, label o indirizzo) è mostrata con lettere minuscole.

Per operandi strettamente correlati, si elenca ogni tipo separatamente senza ripetere il codice mnemonico. Per esempio, ecco alcuni casi del formato di ingresso:

```
                LD  rp,(addr)
                    xy,(addr)
che sono:       LD  SC,(DAT2)
                LD  IX,(MEM)
```

CODICI OGGETTO DELLE ISTRUZIONI

La Tabella 3-4 riporta il codice oggetto e la lunghezza dell'istruzione in byte per ogni tipo d'istruzione. La Tabella 3-5 elenca i codici oggetto in ordine numerico.

Per byte d'istruzione senza variazioni, i codici oggetto sono rappresentati come due digit esadecimali (per es. 3F).

Per byte d'istruzione con variazioni in uno dei due digit, il codice oggetto è mostrato, in Tabella 3-5, come un digit binario a 4 bit ed un digit esadecimale (per es. 11 x 1D). Per altri byte d'istruzione con variazioni, il codice oggetto è mostrato come otto digit binari (per es. 01sss001).

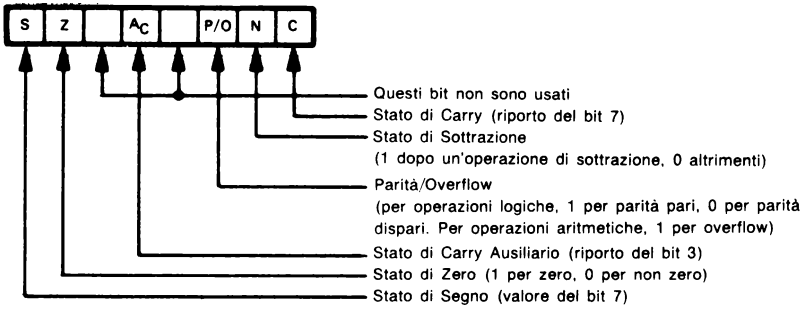
TEMPI DI ESECUZIONE DELLE ISTRUZIONI

La Tabella 3-4 elenca i tempi di esecuzione delle istruzioni in periodi di clock. Si può ottenere il tempo reale dividendo il dato numero di periodi di clock per la velocità del clock. Per esempio, per un'istruzione che richiede 7 periodi di clock, un clock a 4MHz avrà come risultato un tempo di esecuzione di 1,75 microsecondi.

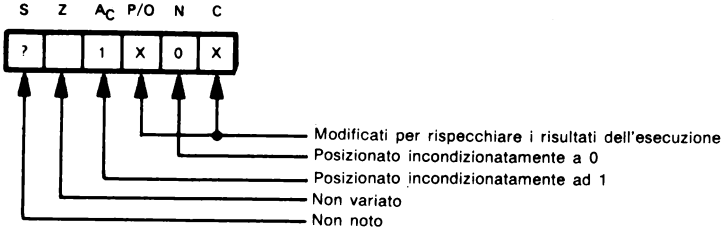
Quando sono mostrati due possibili tempi di esecuzione (cioè 5/11), ciò indica che il numero dei periodi di clock dipende dai flag delle condizioni. Il primo tempo vale per «condizione non verificata», mentre il secondo vale per «condizione verificata».

STATO

I sei flag di stato sono memorizzati nel registro di Flag (F) come segue:



Nelle descrizioni delle singole istruzioni, l'effetto dell'esecuzione dell'istruzione sullo stato è illustrato come segue:



Una X identifica uno stato che è posizionato a 1 o a 0. Uno 0 identifica uno stato che è sempre azzerato. Un 1 identifica uno stato che è sempre posizionato a 1. Un vuoto sta a significare che lo stato non cambia. Una V significa che lo stato è noto.

CAMBIAMENTI DI STATO CON L'ESECUZIONE DI ISTRUZIONI

*Bus Indirizzo: A0-A7: [C]
A8-A15: [B]

Tabella 3-4. Sommario dell'insieme di istruzioni dello Z80

Tipo	Mnemonico	Operando	Codice Oggetto	Byte	Cicli di Clock	Stato C	Stato Z	Stato S	Stato P/O	Stato A_C	Stato N	Operazione Effettuata
I/O	IN	A,(port)	DB yy	2	10							[A] ← [port] Input to Accumulator from directly addressed I/O port. Address Bus: A0-A7: port A8-A15: [A]
	IN	reg,(C)	ED 01ddd000	2	11		X	X	P	X	0	[reg] ← [[C]] Input to register from I/O port addressed by the contents of C.** If second byte is 70_{16} only the flags will be affected.
	INIR		ED B2	2	20/15**		1	?	?	?	1	Repeat until [B] = 0: [[HL]] ← [[C]] [B] ← [B] - 1 [HL] ← [HL] + 1 Transfer a block of data from I/O port addressed by contents of C to memory location addressed by contents of HL, going from low addresses to high. Contents of B serve as a count of bytes remaining to be transferred.**
	INDR		ED BA	2	20/15**		1	?	?	?	1	Repeat until [B] = 0: [[HL]] ← [[C]] [B] ← [B] - 1 [HL] ← [HL] - 1 Transfer a block of data from I/O port addressed by contents of C to memory location addressed by contents of HL, going from high addresses to low. Contents of B serve as a count of bytes remaining to be transferred.**
	INI		ED A2	2	15		X	?	?	?	1	[[HL] ← [[C]] [B] ← [B] - 1 [HL] ← [HL] + 1 Transfer a byte of data from I/O port addressed by contents of C to memory location addressed by contents of HL Decrement byte count and increment destination address.**

*Bus Indirizzo: A0-A7: [C]
A8-A15: [B]

Tabella 3-4. Sommario dell'insieme di istruzioni dello Z80 (segue)

Tipo	Mnemonico	Operando	Codice Oggetto	Byte	Cicli di Clock	Stato C	Stato Z	Stato S	Stato P/O	Stato A_C	Stato N	Operazione Effettuata
I/O (Continua)	IND		ED AA	2	15		X	?	?	?	1	[[HL]] ← [[C]] [B] ← [B] - 1 [HL] ← [HL] - 1 Transfer a byte of data from I/O port addressed by contents of C to memory location addressed by contents of HL. Decrement both byte count and destination address.**
	OUT	(port),A	D3 yy	2	11							[port] ← [A] Output from Accumulator to directly addressed I/O port. Address Bus: A0-A7: port A8-A15: [A]
	OUT	(C),reg	ED 01sss001	2	12							[[C]] ← [reg] Output from register to I/O port addressed by the contents of C.**
	OTIR		ED B3	2	20/15**		1	?	?	?	1	Repeat until [B] = 0: [[C]] ← [[HL]] [B] ← [B] - 1 [HL] ← [HL] + 1 Transfer a block of data from memory location addressed by contents of HL to I/O port addressed by contents of C, going from low memory to high. Contents of B serve as a count of bytes remaining to be transferred.**
	OTDR		ED BB	2	20/15**		1	?	?	?	1	Repeat until [B] = 0: [[C]] ← [[HL]] [B] ← [B] - 1 [HL] ← [HL] - 1 Transfer a block of data from memory location addressed by contents of HL to I/O port addressed by contents of C, going from high memory to low. Contents of B serve as a count of bytes remaining to be transferred.**

*Bus Indirizzo: A0-A7: [C]
A8-A15: [B]

Tabella 3-4. Sommario dell'insieme di istruzioni dello Z80 (segue)

Tipo	Mnemonico	Operando	Codice Oggetto	Byte	Cicli di Clock	Stato C	Stato Z	Stato S	Stato P/O	Stato A_C	Stato N	Operazione Effettuata
I/O (Continua)	OUTI		ED A3	2	15		X	?	?	?	1	[[C]] ← [[HL]] [B] ← [B] - 1 [HL] ← [HL] + 1 Transfer a byte of data from memory location addressed by contents of HL to I/O port addressed by contents of C. Decrement byte count and increment source address.**
	OUTD		ED AB	2	15		X	?	?	?	1	[[C]] ← [[HL]] [B] ← [B] - 1 [HL] ← [HL] - 1 Transfer a byte of data from memory location addressed by contents of HL to I/O port addressed by contents of C. Decrement both byte count and source address.**
Riferimento alla memoria primario	LD	A,(addr)	3A ppqq	3	13							[A] ← [addr] Load Accumulator from directly addressed memory location.
	LD	HL,(addr)	2A ppqq	3	16							[H] ← [addr + 1], [L] ← [addr] Load HL from directly addressed memory.
	LD	rp,(addr) xy,(addr)	ED 01xx1011 ppqq 11x11101 2A ppqq	4 4	20 20							[rp(HI)] ← [addr + 1], [rp(LO)] ← [addr] or [xy(HI)] ← [addr + 1], [xy(LO)] ← [addr] Load register pair or Index register from directly addressed memory.
	LD	(addr),A	32 ppqq	3	13							[addr] ← [A] Store Accumulator contents in directly addressed memory location.
	LD	(addr),HL	22 ppqq	3	16							[addr + 1] ← [H], [addr] ← [L] Store contents of HL to directly addressed memory location.
	LD	(addr),rp (addr),xy	ED 01xx0011 ppqq 11x11101 22 ppqq	4 4	20 20							[addr + 1] ← [rp(HI)], [addr] ← [rp(LO)] or [addr + 1] ← [xy(HI)], [addr] ← [xy(LO)] Store contents of register pair or Index register to directly addressed memory.
	LD	A,(BC) A,(DE)	0A 1A	1 1	7 7							[A] ← [[BC]] or [A] ← [[DE]] Load Accumulator from memory location addressed by the contents of the specified register pair.

Tabella 3-4. Sommario dell'insieme di istruzioni dello Z80 (segue)

Tipo	Mnemonico	Operando	Codice Oggetto	Byte	Cicli di Clock	Stato C	Stato Z	Stato S	Stato P/O	Stato A_C	Stato N	Operazione Effettuata
Riferimento alla memoria primario (continua)	LD	reg,(HL)	01ddd110	1	7							[reg] ← [[HL]] Load register from memory location addressed by contents of HL.
	LD	(BC),A (DE),A	02 12	1 1	7 7							[[BC]] ← [A] or [[DE]] ← [A] Store Accumulator to memory location addressed by the contents of the specified register pair.
	LD	(HL),reg	01110sss	1	7							[[HL]] ← [reg] Store register contents to memory location addressed by the contents of HL.
	LD	reg,(xy+disp)	11x11101 01ddd110 disp	3	19							[reg] ← [[xy] + disp] Load register from memory location using base relative addressing.
	LD	(xy+disp),reg	11x11101 01110sss disp	3	19							[[xy] + disp] ← [reg] Store register to memory location addressed relative to contents of Index register.
Ricerca e trasferimento di un blocco	LDIR		ED B0	2	20/16**				0	0	0	Repeat until [BC] = 0: [[DE]] ← [[HL]] [DE] ← [DE] + 1 [HL] ← [HL] + 1 [BC] ← [BC] - 1 Transfer a block of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE, going from low addresses to high. Contents of BC serve as a count of bytes to be transferred.
	LDDR		ED B8	2	20/16**				0	0	0	Repeat until [BC] = 0: [[DE]] ← [[HL]] [DE] ← [DE] - 1 [HL] ← [HL] - 1 [BC] ← [BC] - 1 Transfer a block of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE, going from high addresses to low. Contents of BC serve as a count of bytes to be transferred.

Tabella 3-4. Sommario dell'insieme di istruzioni dello Z80 (segue)

Tipo	Mnemonico	Operando	Codice Oggetto	Byte	Cicli di Clock	Stato C	Stato Z	Stato S	Stato P/O	Stato A_C	Stato N	Operazione Effettuata
Ricerca e trasferimento di un blocco (Continua)	LDI		ED A0	2	16				X	0	0	[[DE]] ← [[HL]] [DE] ← [DE] + 1 [HL] ← [HL] + 1 [BC] ← [BC] - 1 Transfer one byte of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE. Increment source and destination addresses and decrement byte count.
	LDD		ED A8	2	16				X	0	0	[[DE]] ← [[HL]] [DE] ← [DE] - 1 [HL] ← [HL] - 1 [BC] ← [BC] - 1 Transfer one byte of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE. Decrement source and destination addresses and byte count.
	CPIR		ED B1	2	20/16**		X	X	X	X	1	Repeat until [A] = [[HL]] or [BC] = 0: [A] - [[HL]] (only flags are affected) [HL] ← [HL] + 1 [BC] ← [BC] - 1 Compare contents of Accumulator with those of memory block addressed by contents of HL, going from low addresses to high. Stop when a match is found or when the byte count becomes zero.
	CPDR		ED B9	2	20/16**		X	X	X	X	1	Repeat until [A] = [[HL]] or [BC] = 0: [A] - [[HL]] (only flags are affected) [HL] ← [HL] - 1 [BC] ← [BC] - 1 Compare contents of Accumulator with those of memory block addressed by contents of HL, going from high addresses to low. Stop when a match is found or when the byte count becomes zero.

Table 3-4. A Summary of the Z80 Instruction Set (Continued)

Tipo	Mnemonico	Operando	Codice Oggetto	Byte	Cicli di Clock	Stato C	Stato Z	Stato S	Stato P/O	Stato A_C	Stato N	Operazione Effettuata
Ricerca e trasferimento di un blocco (Continua)	CPI		ED A1	2	16		X	X	X	X	1	[A] - [[HL]] (only flags are affected) [HL] ← [HL] + 1 [BC] ← [BC] - 1 Compare contents of Accumulator with those of memory location addressed by contents of HL. Increment address and decrement byte count.
	CPD		ED A9	2	16		X	X	X	X	1	[A] - [[HL]] (only flags are affected) [HL] ← [HL] - 1 [BC] ← [BC] - 1 Compare contents of Accumulator with those of memory location addressed by contents of HL. Decrement address and byte count.
Riferimento alla memoria secondario	ADD	A,(HL) A,(xy +disp)	86 11x11101 86 disp	1 3	7 19	X	X	X	O	X	0	[A] ← [A] + [[HL]] or [A] ← [A] + [[xy] + disp] Add to Accumulator using implied addressing or base relative addressing.
	ADC	A,(HL) A,(xy +disp)	8E 11x11101 8E disp	1 3	7 19	X	X	X	O	X	0	[A] ← [A] + [[HL]] + C or [A] ← [A] + [[xy] + disp] + C Add with Carry using implied addressing or base relative addressing.
	SUB	(HL) (xy + disp)	96 11x11101 96 disp	1 3	7 19	X	X	X	O	X	1	[A] ← [A] - [[HL]] or [A] ← [A] - [[xy] + disp] Subtract from Accumulator using implied addressing or base relative addressing.
	SBC	A,(HL) A,(xy +disp)	9E 11x11101 9E disp	1 3	7 19	X	X	X	O	X	1	[A] ← [A] - [[HL]] - C or [A] ← [A] - [[xy] + disp] - C Subtract with Carry using implied addressing or base relative addressing.
	AND	(HL) (xy + disp)	A6 11x11101 A6 disp	1 3	7 19	0	X	X	P	1	0	[A] ← [A] Λ [[HL]] or [A] ← [A] Λ [[xy] + disp] AND with Accumulator using implied addressing or base relative addressing.
	OR	(HL) (xy + disp)	B6 11x11101 B6 disp	1 3	7 19	0	X	X	P	1	0	[A] ← [A] V [[HL]] or [A] ← [A] V [[xy] + disp] OR with Accumulator using implied addressing or base relative addressing.

Tabella 3-4. Sommario dell'insieme di istruzioni dello Z80 (segue)

Tipo	Mnemonico	Operando	Codice Oggetto	Byte	Cicli di Clock	Stato C	Stato Z	Stato S	Stato P/O	Stato A_C	Stato N	Operazione Effettuata
Riferimento alla memoria secondario (Continua)	XOR	(HL)	AE	1	7	0	X	X	P	1	0	[A] ← [A] ⊻ [[HL]] or [A] ← [A] ⊻ [[xy] + disp] Exclusive-OR with Accumulator using implied addressing or base relative addressing.
		(xy + disp)	11x11101 AE disp	3	19							
	CP	(HL)	BE	1	7	X	X	X	O	X	1	[A] - [[HL]] or [A] - [[xy] + disp] Compare with Accumulator using implied addressing or base relative addressing. Only the flags are affected.
		(xy + disp)	11x11101 BE disp	3	19							
	INC	(HL)	34	1	11		X	X	O	X	0	[[HL]] ← [[HL]] + 1 or [[xy] + disp] ← [[xy] + disp] + 1 Increment using implied addressing or base relative addressing.
		(xy + disp)	11x11101 34 disp	3	23							
	DEC	(HL)	35	1	11		X	X	O	X	1	[[HL]] ← [[HL]] - 1 or [[xy] + disp] ← [[xy] + disp] - 1 Decrement using implied addressing or base relative addressing.
		(xy + disp)	11x11101 35 disp	3	23							
Rotazione e spostamento di memoria	RLC	(HL)	CB 06	2	15	X	X	X	P	0	0	C ← 7 ← 0 [[HL]] or [[xy] + disp] Rotate contents of memory location (implied or base relative addressing) left with branch Carry.
		(xy + disp)	11x11101 CB disp 06	4	23							
	RL	(HL)	CB 16	2	15	X	X	X	P	0	0	C ← 7 ← 0 [[HL]] or [[xy] + disp] Rotate contents of memory location left through Carry.
		(xy + disp)	11x11101 CB disp 16	4	23							
	RRC	(HL)	CB 0E	2	15	X	X	X	P	0	0	7 → 0 → C [[HL]] or [[xy] + disp] Rotate contents of memory location right with branch Carry.
		(xy + disp)	11x11101 CB disp 0E	4	23							

Tabella 3-4. Sommario dell'insieme di istruzioni dello Z80 (segue)

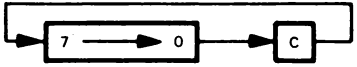
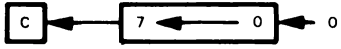
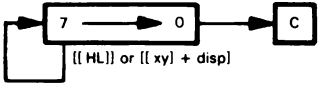
Tipo	Mnemonico	Operando	Codice Oggetto	Byte	Cicli di Clock	Stato C	Stato Z	Stato S	Stato P/O	Stato A_C	Stato N	Operazione Effettuata
Rotazione e spostamento di memoria (continua)	RR	(HL) (xy + disp)	CB 1E 11x11101 CB disp 1E	2 4	15 23	X	X	X	P	0	0	 [[HL]] or [[xy] + disp] Rotate contents of memory location right through Carry
	SLA	(HL) (xy + disp)	CB 26 11x11101 CB disp 26	2 4	15 23	X	X	X	P	0	0	 [[HL]] or [[xy] + disp] Shift contents of memory location left and clear LSB (Arithmetic Shift).
	SRA	(HL) (xy + disp)	CB 2E 11x11101 CB disp 2E	2 4	15 23	X	X	X	P	0	0	 [[HL]] or [[xy] + disp] Shift contents of memory location right and preserve MSB (Arithmetic Shift).
	SRL	(HL) (xy + disp)	CB 3E 11x11101 CB disp 3E	2 4	15 23	X	X	X	P	0	0	 [[HL]] or [[xy] + disp] Shift contents of memory location right and clear MSB (Logical Shift)

Tabella 3-4. Sommario dell'insieme di istruzioni dello Z80 (segue)

Tipo	Mnemonico	Operando	Codice Oggetto	Byte	Cicli di Clock	Stato C	Stato Z	Stato S	Stato P/O	Stato A_C	Stato N	Operazione Effettuata
Immediato	LD	reg,data	00ddd110 yy	2	7							[reg] ← data Load immediate into register.
	LD	rp,data16	00xx0001 yyyy	3	10							[rp] ← data16 or [xy] ← data16
		xy,data16	11x11101 21 yyyy	4	14							Load 16 bits of immediate data into register pair or Index register.
	LD	(HL),data	36 yy	2	10							[[HL]] ← data or [[xy] + disp] ← data
		(xy + disp), data	11x11101 36 disp yy	4	19							Load immediate into memory location using implied or base relative addressing.
Salto	JP	label	C3 ppqq	3	10							[PC] ← label Jump to instruction at address represented by label.
	JR	disp	18 (disp-2)	2	12							[PC] ← [PC] + 2 + (disp-2) Jump relative to present contents of Program Counter.
	JP	(HL)	E9	1	4							[PC] ← [HL] or [PC] ← [xy]
		(xy)	11x11101 E9	2	8							Jump to address contained in HL or Index register.
Ritorno da e chiamata di un sottoprogramma	CALL	label	CD ppqq	3	17							[[SP] - 1] ← [PC(HI)] [[SP] - 2] ← [PC(LO)] [SP] ← [SP] - 2 [PC] ← label Jump to subroutine starting at address represented by label.
	CALL	cond,label	11ccc100 ppqq	3	10/17							Jump to subroutine if condition is satisfied; otherwise, continue in sequence.
	RET		C9	1	10							[PC(LO)] ← [[SP]] [PC(HI)] ← [[SP] + 1] [SP] ← [SP] + 2 Return from subroutine.
	RET	cond	11ccc000	1	5/11							Return from subroutine if condition is satisfied; otherwise, continue in sequence.

Tabella 3-4. Sommario dell'insieme di istruzioni dello Z80 (segue)

Tipo	Mnemonico	Operando	Codice Oggetto	Byte	Cicli di Clock	Stato C	Stato Z	Stato S	Stato P/O	Stato A_C	Stato N	Operazione Effettuata
Operazione immediata	ADD	A,data	C6 yy	2	7	X	X	X	O	X	0	[A] ← [A] + data Add immediate to Accumulator.
	ADC	A,data	CE yy	2	7	X	X	X	O	X	0	[A] ← [A] + data + C Add immediate with Carry.
	SUB	data	D6 yy	2	7	X	X	X	O	X	1	[A] ← [A] - data Subtract immediate from Accumulator.
	SBC	A,data	DE yy	2	7	X	X	X	O	X	1	[A] ← [A] - data - C Subtract immediate with Carry.
	AND	data	E6 yy	2	7	0	X	X	P	1	0	[A] ← [A] ∧ data AND immediate with Accumulator.
	OR	data	F6 yy	2	7	0	X	X	P	1	0	[A] ← [A] V data OR immediate with Accumulator.
	XOR	data	EE yy	2	7	0	X	X	P	1	0	[A] ← [A] ⊻ data Exclusive-OR immediate with Accumulator.
	CP	data	FE yy	2	7	X	X	X	O	X	1	[A] - data Compare immediate data with Accumulator contents; only the flags are affected.
Salto su condizione	JP	cond,label	11ccc010 ppqq	3	10							If cond, then [PC] ← label Jump to instruction at address represented by label if the condition is true.
	JR	C,disp	38 (disp-2)	2	7/12							If C = 1, then [PC] ← [PC] + 2 + (disp − 2) Jump relative to contents of Program Counter if Carry flag is set.
	JR	NC,disp	30 (disp-2)	2	7/12							If C = 0, then [PC] ← [PC] + 2 + (disp −2) Jump relative to contents of Program Counter if Carry flag is reset.
	JR	Z,disp	28 (disp-2)	2	7/12							If Z = 1, then [PC] ← [PC] + 2 + (disp −2) Jump relative to contents of Program Counter if Zero flag is set.
	JR	NZ,disp	20 (disp-2)	2	7/12							If Z = 0, then [PC] ← [PC] + 2 + (disp −2) Jump relative to contents of Program Counter if Zero flag is reset.
	DJNZ	disp	10 (disp-2)	2	8/13							[B] ← [B] - 1 If [B] ≠ 0, then [PC] + 2 + (disp −2) Decrement contents of B and Jump relative to contents of Program Counter if result is not 0.

Tabella 3-4. Sommario dell'insieme di istruzioni dello Z80 (segue)

Tipo	Mnemonico	Operando	Codice Oggetto	Byte	Cicli di Clock	Stato C	Stato Z	Stato S	Stato P/O	Stato A_C	Stato N	Operazione Effettuata
Spostamento da registro a registro	LD	dst,src	01dddsss	1	4							[dst] ← [src] Move contents of source register to destination register. Register designations src and dst may each be A, B, C, D, E, H or L.
	LD	A,I	ED 57	2	9		X	X	I	0	0	[A] ← [I] Move contents of Interrupt Vector register to Accumulator.
	LD	A,R	ED 5F	2	9		X	X	I	0	0	[A] ← [R] Move contents of Refresh register to Accumulator.
	LD	I,A	ED 47	2	9							[I] ← [A] Load Interrupt Vector register from Accumulator.
	LD	R,A	ED 4F	2	9							[R] ← [A] Load Refresh register from Accumulator.
	LD	SP,HL	F9	1	6							[SP] ← [HL] Move contents of HL to Stack Pointer.
	LD	SP,xy	11x11101 F9	2	10							[SP] ← [xy] Move contents of Index register to Stack Pointer.
	EX	DE,HL	EB	1	4							[DE] ← → [HL] Exchange contents of DE and HL.
	EX	AF,AF'	08	1	4							[AF] ← → [AF'] Exchange program status and alternate program status.
	EXX		D9	1	4							$\begin{pmatrix}[BC]\\[DE]\\[HL]\end{pmatrix} \leftarrow\rightarrow \begin{pmatrix}[BC']\\[DE']\\[HL']\end{pmatrix}$ Exchange register pairs and alternate register pairs.

Tabella 3-4. Sommario dell'insieme di istruzioni dello Z80 (segue)

Tipo	Mnemonico	Operando	Codice Oggetto	Byte	Cicli di Clock	Stato C	Stato Z	Stato S	Stato P/O	Stato A_C	Stato N	Operazione Effettuata
Operazione tra registro e registro	ADD	A,reg	10000rrr	1	4	X	X	X	O	X	0	[A] ← [A] + [reg] Add contents of register to Accumulator.
	ADC	A,reg	10001rrr	1	4	X	X	X	O	X	0	[A] ← [A] + [reg] + C Add contents of register and Carry to Accumulator.
	SUB	reg	10010rrr	1	4	X	X	X	O	X	1	[A] ← [A] - [reg] Subtract contents of register from Accumulator.
	SBC	A,reg	10011rrr	1	4	X	X	X	O	X	1	[A] ← [A] - [reg] - C Subtract contents of register and Carry from Accumulator.
	AND	reg	10000rrr	1	4	0	X	X	P	1	0	[A] ← [A] ∧ [reg] AND contents of register with contents of Accumulator.
	OR	reg	10110rrr	1	4	0	X	X	P	1	0	[A] ← [A] V [reg] OR contents of register with contents of Accumulator.
	XOR	reg	10101rrr	1	4	0	X	X	P	1	0	[A] ← [A] ⊻ [reg] Exclusive-OR contents of register with contents of Accumulator.
	CP	reg	10111rrr	1	4	X	X	X	O	X	1	[A] - [reg] Compare contents of register with contents of Accumulator. Only the flags are affected.
	ADD	HL,rp	00xx1001	1	11	X				?	0	[HL] ← [HL] + [rp] 16-bit add register pair contents to contents of HL.
	ADC	HL,rp	ED 01xx1010	2	15	X	X	X	O	?	0	[HL] ← [HL] + [rp] + C 16-bit add with Carry register pair contents to contents of HL.
	SBC	HL,rp	ED 01xx0010	2	15	X	X	X	O	?	1	[HL] ← [HL] - [rp] - C 16-bit subtract with Carry register pair contents from contents of HL.
	ADD	IX,pp	DD 00xx1001	2	15	X				?	0	[IX] ← [IX] + [pp] 16-bit add register pair contents to contents of Index register IX (pp = BC, DE, IX, SP).
	ADD	IY,rr	FD 00xx1001	2	15	X				?	0	[IY] ← [IY] + [rr] 16-bit add register pair contents to contents of Index register IY (rr = BC, DE, IY, SP).

Tabella 3-4. Sommario dell'insieme di istruzioni dello Z80 (segue)

Tipo	Mnemonico	Operando	Codice Oggetto	Byte	Cicli di Clock	Stato C	Stato Z	Stato S	Stato P/O	Stato A_C	Stato N	Operazione Effettuata
Operazione di registri	DAA		27	1	4	X	X	X	P	X		Decimal adjust Accumulator, assuming that Accumulator contents are the sum or difference of BCD operands.
	CPL		2F	1	4					1	1	[A] ← [$\overline{A}$] Complement Accumulator (ones complement).
	NEG		ED 44	2	8	X	X	X	O	X	1	[A] ← [$\overline{A}$] + 1 Negate Accumulator (twos complement).
	INC	reg	00rrr100	1	4		X	X	O	X	0	[reg] ← [reg] + 1 Increment register contents.
	INC	rp xy	00xx0011 11x11101 23	1 2	6 10							[rp] ← [rp] + 1 or [xy] ← [xy] + 1 Increment contents of register or Index register.
	DEC	reg	00rrr101	1	4		X	X	O	X	1	[reg] ← [reg] - 1 Decrement register contents.
	DEC	rp xy	00xx1011 11x11101 2B	1 2	6 10							[rp] ← [rp] - 1 or [xy] ← [xy] - 1 Decrement contents of register pair or Index register.
Rotazione e spostamento di registri	RLCA		07	1	4	X				0	C	C ← 7 ← 0 [A] Rotate Accumulator left with branch Carry.
	RLA		17	1	4	X				0	0	C ← 7 ← 0 [A] Rotate Accumulator left through Carry.
	RRCA		0F	1	4	X				0	0	7 → 0 → C [A] Rotate Accumulator right with branch Carry.

Tabella 3-4. Sommario dell'insieme di istruzioni dello Z80 (segue)

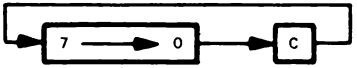
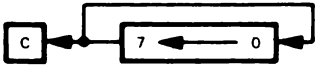
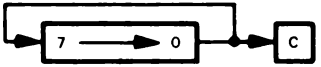
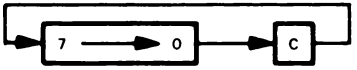
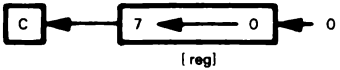
Tipo	Mnemonico	Operando	Codice Oggetto	Byte	Cicli di Clock	Stato C	Stato Z	Stato S	Stato P/O	Stato A_C	Stato N	Operazione Effettuata
Rotazione e spostamento di registri (Continua)	RRA		1F	1	4	X				0	0	 [A] Rotate Accumulator right through Carry.
	RLC	reg	CB 00000rrr	2	8	X	X	X	P	0	0	 [reg] Rotate contents of register left with branch Carry.
	RL	reg	CB 00010rrr	2	8	X	X	X	P	0	0	[reg] Rotate contents of register left through Carry.
	RRC	reg	CB 00001rrr	2	8	X	X	X	P	0	0	 [reg] Rotate contents of register right with branch Carry.
	RR	reg	CB 00011rrr	2	8	X	X	X	P	0	0	 [reg] Rotate contents of register right through Carry.
	SLA	reg	CB 00100rrr	2	8	X	X	X	P	0	0	 [reg] Shift contents of register left and clear LSB (Arithmetic Shift).

Tabella 3-4. Sommario dell'insieme di istruzioni dello Z80 (segue)

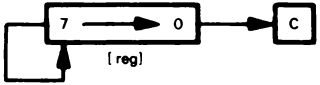
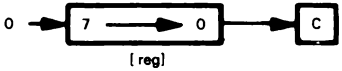
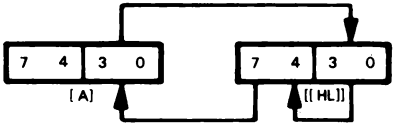
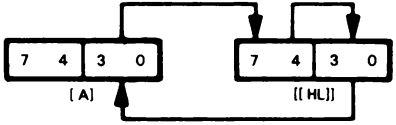
Tipo	Mnemonico	Operando	Codice Oggetto	Byte	Cicli di Clock	Stato C	Stato Z	Stato S	Stato P/O	Stato A_C	Stato N	Operazione Effettuata
Rotazione e spostamento di registri (Continua)	SRA	reg	CB 00101rrr	2	8	X	X	X	P	0	0	 Shift contents of register right and preserve MSB (Arithmetic Shift).
	SRL	reg	CB 00111rrr	2	8	X	X	X	P	0	0	 Shift contents of register right and clear MSB (Logical Shift).
	RLD		ED 6F	2	18		X	X	P	0	0	 Rotate one BCD digit left between the Accumulator and memory location (implied addressing).Contents of the upper half of the Accumulator are not affected.
	RRD		ED 67	2	18		X	X	P	0	0	 Rotate one BCD digit right between the Accumulator and memory location (implied addressing).Contents of the upper half of the Accumulator are not affected.

Tabella 3-4. Sommario dell'insieme di istruzioni dello Z80 (segue)

Tipo	Mnemonico	Operando	Codice Oggetto	Byte	Cicli di Clock	Stato C	Stato Z	Stato S	Stato P/O	Stato A_C	Stato N	Operazione Effettuata
Manipolazione di bit	BIT	b,reg	CB 01bbbrrr	2	8		X	?	?	1	0	Z ← $\overline{reg(b)}$ Zero flag contains complement of the selected register bit.
	BIT	b,(HL) b,(xy + disp)	CB 01bbb110 11x11101 CB disp 01bbb110	2 4	12 20		X	?	?	1	0	Z ← $\overline{[[HL]](b)}$ or Z ← $\overline{[[xy] + disp](b)}$ Zero flag contains complement of selected bit of the memory location (implied addressing or base relative addressing).
	SET	b,reg	CB 11bbbrrr	2	8							reg(b) ← 1 Set indicated register bit.
	SET	b,(HL) b,(xy + disp)	CB 11bbb110 11x11101 CB disp 11bbb110	2 4	15 23							[[HL]](b) ← 1 or [[xy] + disp](b) ← 1 Set indicated bit of memory location (implied addressing or base relative addressing).
	RES	b,reg	CB 10bbbrrr	2	8							reg(b) ← 0 Reset indicated register bit.
	RES	b,(HL) b,(xy + disp)	CB 10bbb110 11x11101 CB disp 10bbb110	2 4	15 23							[[HL]](b) ← 0 or [[xy] + disp](b) ← 0 Reset indicated bit in memory location (implied addressing or base relative addressing).
Stack	PUSH	pr xy	11xx0101 11x11101 E5	1 2	11 15							[[SP]-1] ← [pr(HI)] [[SP]-2] ← [pr(LO)] [SP] ← [SP]-2 Put contents of register pair or Index register on top of Stack and decrement Stack Pointer.
	POP	pr xy	11xx0001 11x11101 E1	1 2	10 14							[pr(LO)] ← [[SP]] [pr(HI)] ← [[SP] + 1] [SP] ← [SP] + 2 Put contents of top of Stack in register pair or Index register and increment Stack Pointer.
	EX	(SP),HL (SP),xy	E3 11x11101 E3	1 2	19 23							[H] ←→ [[SP] + 1] [L] ←→ [[SP]] Exchange contents of HL or Index register and top of Stack.

Tabella 3-4. Sommario dell'insieme di istruzioni dello Z80 (segue)

Tipo	Mnemonico	Operando	Codice Oggetto	Byte	Cicli di Clock	Stato C	Stato Z	Stato S	Stato P/O	Stato A_C	Stato N	Operazione Effettuata
Interruzione	DI		F3	1	4							Disable interrupts.
	EI		FB	1	4							Enable interrupts.
	RST.	n	11xxx111	1	11							[[SP]-1] ← [PC(HI)] [[SP]-2] ← [PC(LO)] [SP] ← [SP]-2 [PC] ← $(8 \cdot n)_{16}$ Restart at designated location.
	RETI		ED 4D	2	14							Return from interrupt.
	RETN		ED 45	2	14							Return from nonmaskable interrupt.
	IM	0	ED 46	2	8							Set interrupt mode 0, 1, or 2.
		1	ED 56	2	8							
		2	ED 5E	2	8							
Stato	SCF		37	1	4	1				0	0	C ← 1 Set Carry flag.
	CCF		3F	1	4	X				?	0	C ← $\overline{C}$ Complement Carry flag.
	NOP		00	1	4							No operation — volatile memories are refreshed.
	HALT		76	1	4							CPU halts, executes NOPs to refresh volatile memories.

••I tempi di esecuzione indicati valgono per una interazione.

Tabella 3-5. Codici Oggetto delle istruzioni in ordine numerico

Codice Oggetto	Istruzione	
00	NOP	
01 yyyy	LD	BC,data16
02	LD	(BC),A
03	INC	BC
04	INC	B
05	DEC	B
06 yy	LD	B,data
07	RLCA	
08	EX	AF,AF'
09	ADD	HL,BC
0A	LD	A,(BC)
0B	DEC	BC
0C	INC	C
0D	DEC	C
0E yy	LD	C,data
0F	RRCA	
10 disp-2	DJNZ	disp
11 yyyy	LD	DE,data16
12	LD	(DE),A
13	INC	DE
14	INC	D
15	DEC	D
16 yy	LD	D,data
17	RLA	
18 disp-2	JR	disp
19	ADD	HL,DE
1A	LD	A,(DE)
1B	DEC	DE
1C	INC	E
1D	DEC	E
1E yy	LD	E,data
1F	RRA	
20 disp-2	JR	NZ,disp
21 yyyy	LD	HL,data16
22 ppqq	LD	(addr),HL
23	INC	HL
24	INC	H
25	DEC	H
26 yy	LD	H,data
27	DAA	
28 disp-2	JR	Z,disp
29	ADD	HL,HL
2A ppqq	LD	HL,(addr)
2B	DEC	HL
2C	INC	L
2D	DEC	L
2E	LD	L,data
2F	CPL	
30 disp-2	JR	NC,disp
31 yyyy	LD	SP,data16
32 ppqq	LD	(addr),A
33	INC	SP
34	INC	(HL)
35	DEC	(HL)
36 yy	LD	(HL),data
37	SCF	
38	JR	C,disp

Codice Oggetto	Istruzione	
39	ADD	HL,SP
3A ppqq	LD	A,(addr)
3B	DEC	SP
3C	INC	A
3D	DEC	A
3E yy	LD	A,data
3F	CCF	
4 0sss	LD	B,reg
46	LD	B,(HL)
4 1sss	LD	C,reg
4E	LD	C,(HL)
5 0sss	LD	D,reg
56	LD	D,(HL)
5 1sss	LD	E,reg
5E	LD	E,(HL)
6 0sss	LD	H,reg
66	LD	H,(HL)
6 1sss	LD	L,reg
6E	LD	L,(HL)
7 0sss	LD	(HL),reg
76	HALT	
7 1sss	LD	A,reg
7E	LD	A,(HL)
8 0rrr	ADD	A,reg
86	ADD	A,(HL)
8 1rrr	ADC	A,reg
8E	ADC	A,(HL)
9 0rrr	SUB	reg
96	SUB	(HL)
9 1rrr	SBC	A,reg
9E	SBC	A,(HL)
A 0rrr	AND	reg
A6	AND	(HL)
A 1rrr	XOR	reg
AE	XOR	(HL)
B 0rrr	OR	reg
B6	OR	(HL)
B 1rrr	CP	reg
BE	CP	(HL)
C0	RET	NZ
C1	POP	BC
C2 ppqq	JP	NZ,addr
C3 ppqq	JP	addr
C4 ppqq	CALL	NZ,addr
C5	PUSH	BC
C6 yy	ADD	A,data
C7	RST	00H
C8	RET	Z
C9	RET	
CA ppqq	JP	Z,addr
CB 0 0rrr	RLC	reg
CB 06	RLC	(HL)
CB 0 1rrr	RRC	reg
CB 0E	RRC	(HL)
CB 1 0rrr	RL	reg
CB 16	RL	(HL)
CB 1 1rrr	RR	reg

Tabella 3-5. Codici Oggetto delle Istruzioni in ordine numerico (segue)

Codice Oggetto	Istruzione	
CB 1E	RR	(HL)
CB 2 0rrr	SLA	reg
CB 26	SLA	(HL)
CB 2 1rrr	SRA	reg
CB 2E	SRA	(HL)
CB 3 1rrr	SRL	reg
CB 3E	SRL	(HL)
CB 01bbbrrr	BIT	b,reg
CB 01bbb110	BIT	b,(HL)
CB 10bbbrrr	RES	b,reg
CB 10bbb110	RES	b,(HL)
CB 11bbbrrr	SET	b,reg
CB 11bbb110	SET	b,(HL)
CC ppqq	CALL	Z,addr
CD ppqq	CALL	addr
CE yy	ADC	A,data
CF	RST	08H
D0	RET	NC
D1	POP	DE
D2 ppqq	JP	NC,addr
D3 yy	OUT	(port),A
D4 ppqq	CALL	NC,addr
D5	PUSH	DE
D6 yy	SUB	data
D7	RST	10H
D8	RET	C
D9	EXX	
DA ppqq	JP	C,addr
DB yy	IN	A,(port)
DC ppqq	CALL	C,addr
DD 00xx 9	ADD	IX,pp
DD 21 yyyy	LD	IX,data16
DD 22 ppqq	LD	(addr),IX
DD 23	INC	IX
DD 2A ppqq	LD	IX,(addr)
DD 2B	DEC	IX
DD 34 disp	INC	(IX + disp)
DD 35 disp	DEC	(IX + disp)
DD 36 disp yy	LD	(IX + disp),data
DD 01ddd110 disp	LD	reg,(IX + disp)
DD 1 0sss disp	LD	(IX + disp),reg
DD 86 disp	ADD	A,(IX + disp)
DD 8E disp	ADC	A,(IX + disp)
DD 96 disp	SUB	(IX + disp)
DD 9E disp	SBC	A,(IX + disp)
DD A6 disp	AND	(IX + disp)
DD AE disp	XOR	(IX + disp)
DD B6 disp	OR	(IX + disp)
DD BE disp	CP	(IX + disp)
DD CB disp 06	RLC	(IX + disp)
DD CB disp 0E	RRC	(IX + disp)
DD CB disp 16	RL	(IX + disp)
DD CB disp 1E	RR	(IX + disp)
DD CB disp 26	SLA	(IX + disp)
DD CB disp 2E	SRA	(IX + disp)
DD CB disp 3E	SRL	(IX + disp)
DD CB disp 01bbb110	BIT	b,(IX + disp)

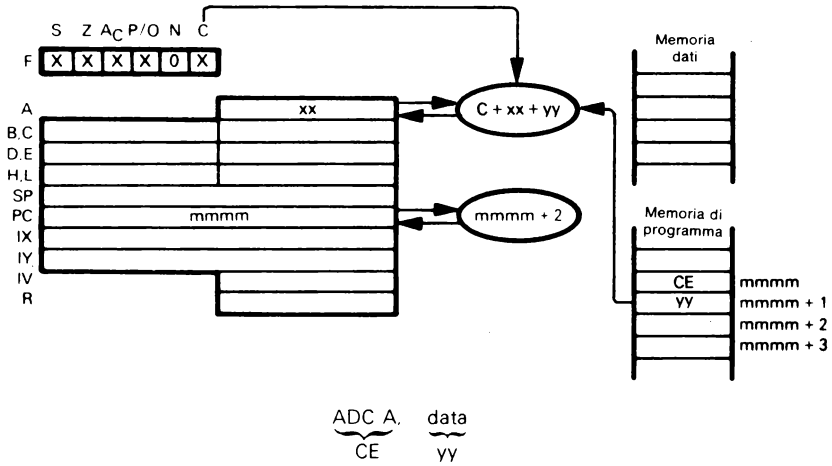
Codice Oggetto	Istruzione	
DD CB disp 10bbb110	RES	b,(IX + disp)
DD CB disp 11bbb110	SET	b,(IX + disp)
DD E1	POP	IX
DD E3	EX	(SP),IX
DD E5	PUSH	IX
DD E9	JP	(IX)
DD F9	LD	SP,IX
DE yy	SBC	A,data
DF	RST	18H
E0	RET	PO
E1	POP	HL
E2 ppqq	JP	PO,addr
E3	EX	(SP),HL
E4 ppqq	CALL	PO,addr
E5	PUSH	HL
E6 yy	AND	data
E7	RST	20H
E8	RET	PE
E9	JP	(HL)
EA ppqq	JP	PE,addr
EB	EX	DE,HL
EC ppqq	CALL	PE,addr
ED 01ddd000	IN	reg,(C)
ED 01sss001	OUT	(C),reg
ED 01xx 2	SBC	HL,rp
ED 01xx 3 ppqq	LD	(addr),rp
ED 44	NEG	
ED 45	RETN	
ED 010nn110	IM	m
ED 47	LD	I,A
ED 01xx A	ADC	HL,rp
ED 01xx B ppqq	LD	rp,(addr)
ED 4D	RETI	
ED 4F	LD	R,A
ED 57	LD	A,I
ED 5F	LD	A,R
ED 67	RRD	
ED 6F	RLD	
ED A0	LDI	
ED A1	CPI	
ED A2	INI	
ED A3	OUTI	
ED A8	LDD	
ED A9	CPD	
ED AA	IND	
ED AB	OUTD	
ED B0	LDIR	
ED B1	CPIR	
ED B2	INIR	
ED B3	OTIR	
ED B8	LDDR	
ED B9	CPDR	
ED BA	INDR	
ED BB	OTDR	
EE yy	XOR	data
EF	RST	28H

Tabella 3-5. Codici Oggetto delle Istruzioni in ordine numerico (segue)

Codice Oggetto	Istruzione	
F0	RET	P
F1	POP	AF
F2 ppqq	JP	P,addr
F3	DI	
F4 ppqq	CALL	P,addr
F5	PUSH	AF
F6 yy	OR	data
F7	RST	30H
F8	RET	M
F9	LD	SP,HL
FA ppqq	JP	M,addr
FB	EI	
FC ppqq	CALL	M,addr
FD 00xx 9	ADD	IY,rr
FD 21 yyyy	LD	IY,data16
FD 22 ppqq	LD	(addr),IY
FD 23	INC	IY
FD 2A ppqq	LD	IY,(addr)
FD 2B	DEC	IY
FD 34 disp	INC	(IY + disp)
FD 35 disp	DEC	(IY + disp)
FD 36 disp yy	LD	(IY + disp),data
FD 01ddd110 disp	LD	reg,(IY + disp)
FD 7 0sss disp	LD	(IY + disp),reg
FD 86 disp	ADD	A,(IY + disp)

Codice Oggetto	Istruzione	
FD 8E disp	ADC	A,(IY + disp)
FD 96 disp	SUB	(IY + disp)
FD 9E disp	SBC	A,(IY + disp)
FD A6 disp	AND	(IY + disp)
FD AE disp	XOR	(IY + disp)
FD B6 disp	OR	(IY + disp)
FD BE disp	CP	(IY + disp)
FD CB disp 06	RLC	(IY + disp)
FD CB disp 0E	RRC	(IY + disp)
FD CB disp 16	RL	(IY + disp)
FD CB disp 1E	RR	(IY + disp)
FD CB disp 26	SLA	(IY + disp)
FD CB disp 2E	SRA	(IY + disp)
FD CB disp 3E	SRL	(IY + disp)
FD CB disp 01bbb110	BIT	b,(IY + disp)
FD CB disp 10bbb110	RES	b,(IY + disp)
FD CB disp 11bbb110	SET	b,(IY + disp)
FD E1	POP	IY
FD E3	EX	(SP),IY
FD E5	PUSH	IY
FD E9	JP	(IY)
FD F9	LD	SP,IY
FE yy	CP	data
FF	RST	38H

ADC, A,data – SOMMA IMMEDIATA CON CARRY ALL'ACCUMULATORE

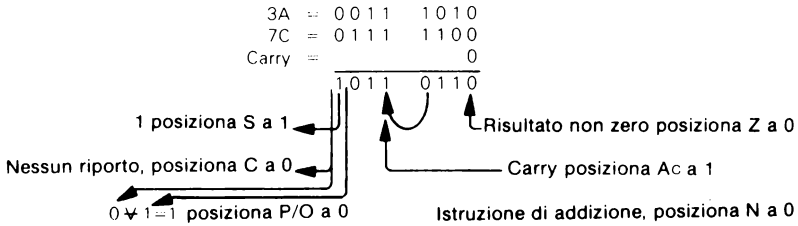


Somma il contenuto del prossimo byte di memoria programmi e lo stato di Carry all'Accumulatore.

Supponiamo che xx = $3A_{16}$, yy = $7C_{16}$ e Carry = 0. Dopo l'esecuzione dell'istruzione:

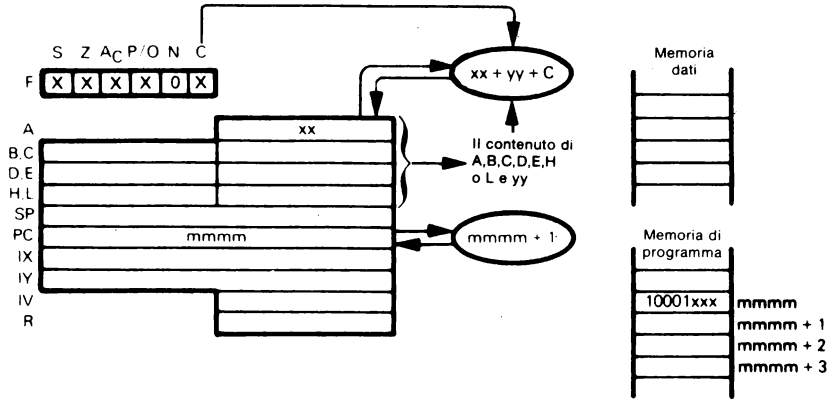
ADC A,7CH

l'Accumulatore conterrà $B6_{16}$:



L'istruzione ADC è frequentemente usata in una somma di più byte per il secondo e i byte seguenti.

ADC A,reg — SOMMA IL REGISTRO CON CARRY ALL'ACCUMULATORE



ADC A, reg

10001 xxx

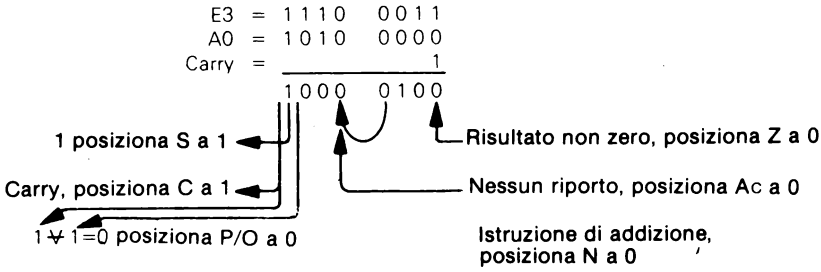
000 per reg = B
001 per reg = C
010 per reg = D
011 per reg = E
100 per reg = H
101 per reg = L
111 per reg = A

Somma il contenuto del Registro A, B, C, D, E, H o L e lo stato di Carry all'Accumulatore.

Supponiamo che xx = $E3_{16}$, che il Registro E contenga $A0_{16}$ e che il Carry = 1. Dopo l'esecuzione dell'istruzione

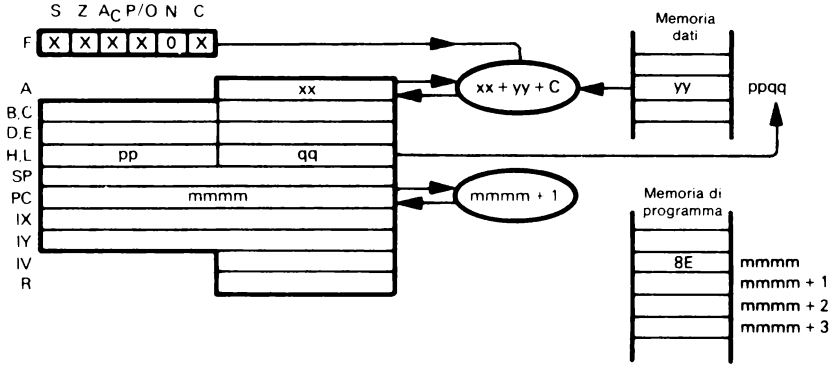
ADC A,E

l'Accumulatore conterrà 84_{16}:



L'istruzione ADC è grandemente usata in una somma a più byte per il secondo ed i byte successivi.

ADC A,(HL) – SOMMA DELLA MEMORIA E DEL CARRY
ADC A,(IX + disp) ALL'ACCUMULATORE
ADC A,(IY + disp)



L'illustrazione mostra l'esecuzione di ADC A,(HL):

$$\underbrace{\text{ADC A,(HL)}}_{\text{8E}}$$

Somma il contenuto della locazione di memoria (specificata dal contenuto della coppia di registri H ed L) e lo stato del Carry all'Accumulatore.

Supponiamo che xx = $E3_{16}$, yy = $A0_{16}$ e Carry = 1. Dopo l'esecuzione dell'istruzione

ADC A,(HL)

l'Accumulatore conterrà 84_{16}:

```
E3    =  1 1 1 0   0 0 1 1
A0    =  1 0 1 0   0 0 0 0
Carry =                  1
         1 0 0 0   0 1 0 0
```

1 posiziona S a 1

Risultato non zero, posiziona Z a 0

Carry posiziona C a 1

Nessun riporto, posiziona Ac a 0

1 ∀ 1=0, posiziona P/O a 0

Istruzione di addizione, posiziona N a 0

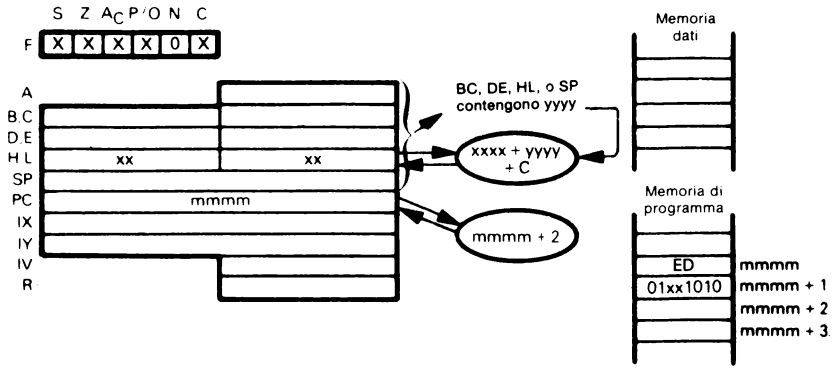
$$\underbrace{\text{ADC A,(IX}}_{\text{DD 8E}}\underbrace{\text{+disp)}}_{\text{d}}$$

Somma il contenuto della locazione di memoria (specificata dalla somma del contenuto del registro IX e del digit d di dislocamento) e del Carry all'Accumulatore.

$$\underbrace{\text{ADC A,(IY}}_{\text{FD 8E}}\underbrace{\text{+disp)}}_{\text{d}}$$

Questa istruzione è identica a ADC A,(IX + disp), tranne che essa usa il registro IY invece del registro IX. L'istruzione ADC è grandemente usata nelle somme con più byte per il secondo e i byte successivi.

ADC HL,rp — SOMMA LA COPPIA DI REGISTRI CON CARRY AD H ED L



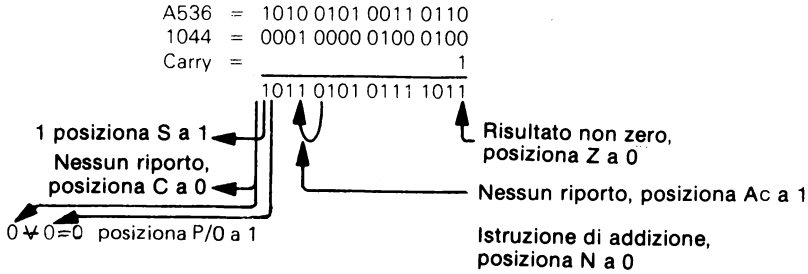
00 per rp è la coppia di registri BC
01 per rp è la coppia di registri DE
10 per rp è la coppia di registri HL
11 per rp è lo Stack Pointer

Somma il valore di 16 bit di una delle coppie di registri BC, DE, HL o dello Stack Pointer e lo stato di Carry alla coppia di registri H ed L.

Supponiamo che HL contenga $A536_{16}$, che BC contenga 1044_{16} e che Carry = 1. Dopo l'esecuzione dell'istruzione

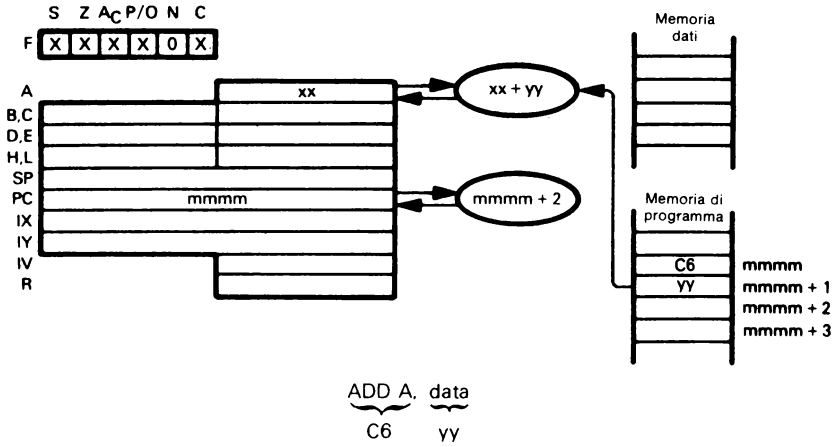
ADC HL,BC

La coppia di registri H ed L conterrà:



L'istruzione ADC è grandemente usata in somme con più byte per il secondo e i byte successivi.

ADD A,data — SOMMA IMMEDIATA ALL'ACCUMULATORE

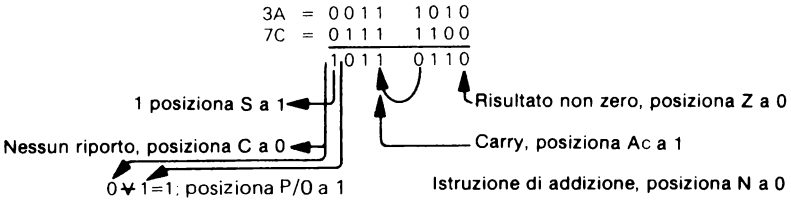


Somma il contenuto del successivo byte della memoria di programma all'Accumulatore.

Supponiamo che xx = $3A_{16}$, yy = $7C_{16}$ e Carry = 0. Dopo l'esecuzione dell'istruzione

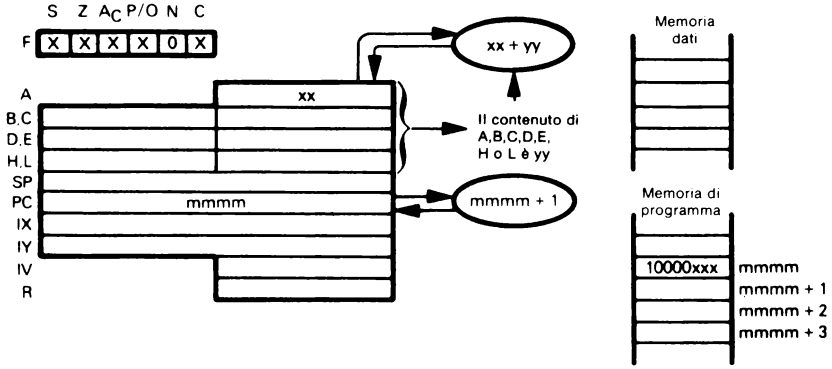
ADD A,7CH

l'Accumulatore conterrà $B6_{16}$:



Questa è un'istruzione ordinaria di manipolazione dei dati.

ADD A,reg — SOMMA IL CONTENUTO DEL REGISTRO ALL'ACCUMULATORE



ADD reg

10000 xxx

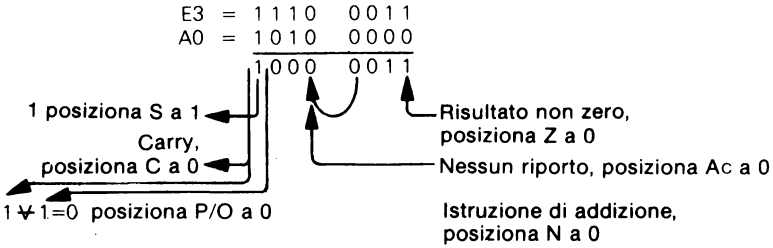
000 per reg =B
001 per reg =C
010 per reg =D
011 per reg =E
100 per reg =H
101 per reg =L
111 per reg =A

Somma il contenuto del Registro A, B, C, D, E, H o L all'Accumulatore.

Supponiamo che xx = $E3_{16}$. Il Registro E contenga $A0_{16}$. Dopo l'esecuzione di

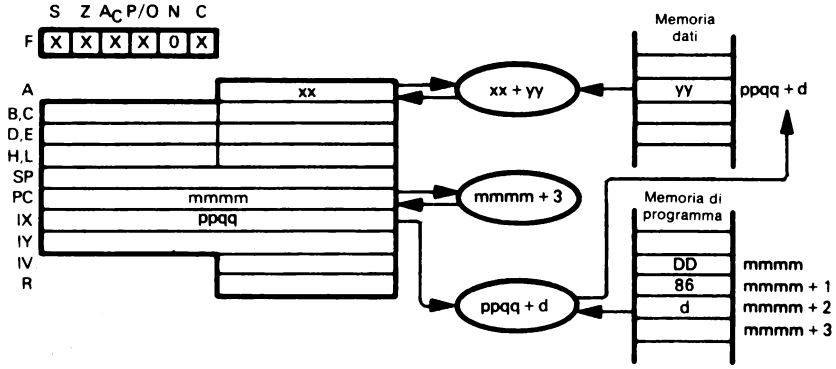
ADD A,E

l'Accumulatore conterrà 83_{16}:



Questa è un'istruzione ordinaria di manipolazione dei dati.

ADD A,(HL) – SOMMA LA MEMORIA ALL'ACCUMULATORE
ADD A,(IX + disp)
ADD A,(IY + disp)



L'illustrazione mostra l'esecuzione di ADD A,(IX + disp).

$$\underbrace{\text{ADD A,(IX+disp)}}$$

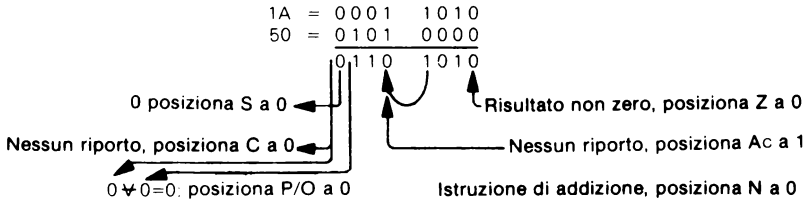
DD 86 d

Somma il contenuto della locazione di memoria (specificata dalla somma del contenuto del registro IX e del digit di dislocamento d) al contenuto dell'Accumulatore.

Supponiamo che ppqq = 4000_{16}, xx = $1A_{16}$, e che la locazione di memoria $400F_{16}$ contenga 50_{16}. Dopo l'esecuzione dell'istruzione

ADD A,(IX + 0FH)

l'Accumulatore conterrà $6A_{16}$.



$$\underbrace{\text{ADD A,(IY+disp)}}$$

FD 86 d

L'istruzione è identica a ADD A,(IX + disp), tranne che essa usa il registro IY invece del registro IX.

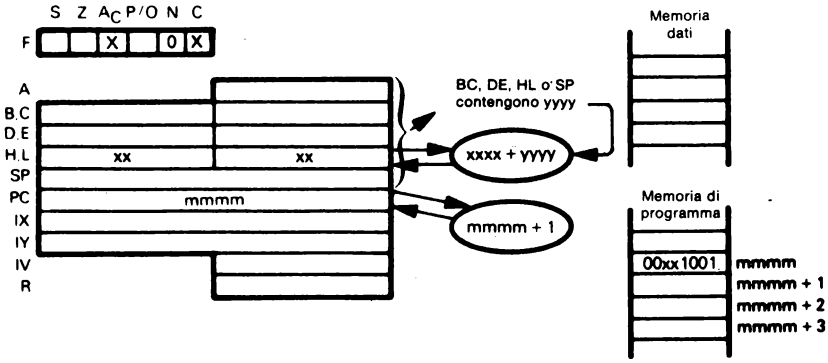
$$\underbrace{\text{ADD A,(HL)}}$$

86

Questa versione dell'istruzione somma il contenuto della locazione di memoria, specificata dal contenuto della coppia di registri H ed L, all'Accumulatore.

L'istruzione ADD è un'istruzione ordinaria di manipolazione dei dati.

ADD HL,rp — SOMMA LA COPPIA DI REGISTRI AD H ED L



00 per rp è la coppia di registri BC
01 per rp è la coppia di registri DE
10 per rp è la coppia di registri HL
11 per rp è lo Stack Pointer

Somma il valore a 16 bit da una delle coppie di registri BC, DE, HL o dallo Stack Pointer alla coppia di registri H ed L.

Supponiamo che HL contenga $034A_{16}$ e che BC contenga $214C_{16}$. Dopo l'esecuzione dell'istruzione

ADD HL,BC

la coppia di registri HL conterrà 2496_{16}.

```
034A  =  0000 0011 0100 1010
214C  =  0010 0001 0100 1100
         ---------------------
         0010 0100 1001 0110
```

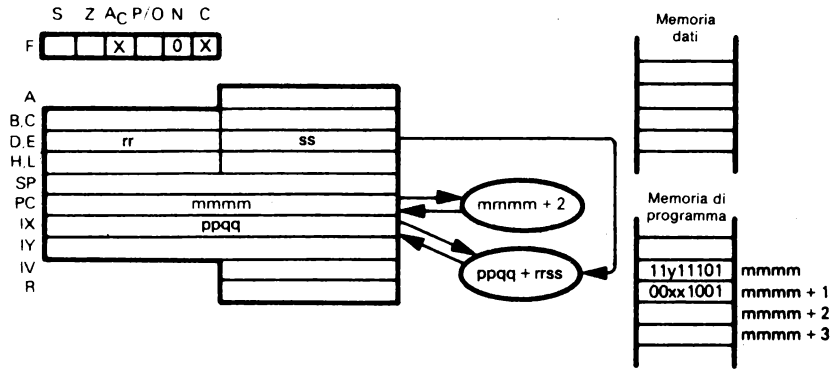
Nessun riporto, posiziona Ac a 0

Nessun riporto, posiziona Ac a 0

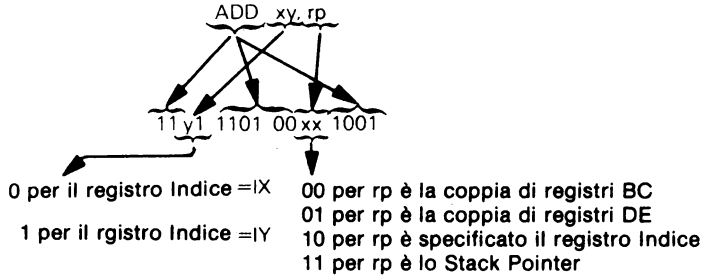
Istruzione di somma, posiziona N a 0

L'istruzione ADD HL,HL è equivalente a uno spostamento a sinistra dei 16 bit.

ADD xy,rp — SOMMA LA COPPIA DI REGISTRI AL REGISTRO INDICE



L'illustrazione mostra l'esecuzione di ADD IX,DE.



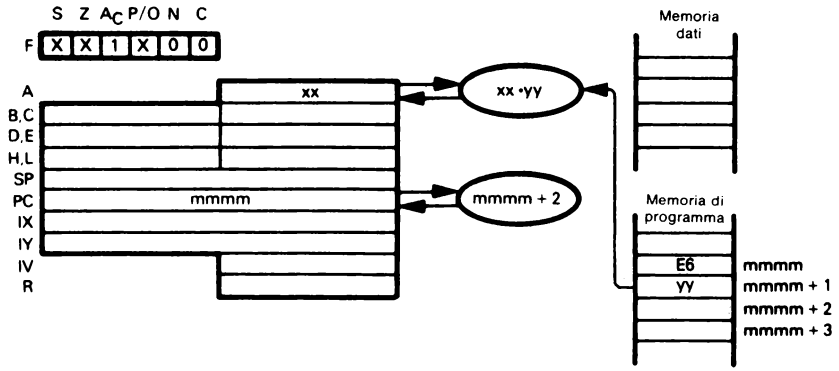
Somma il contenuto della coppia di registri specificata al contenuto del registro Indice specificato.

Supponiamo che IY contenga $4FF0_{16}$ e che BC contenga $000F_{16}$. Dopo l'esecuzione della istruzione

ADD IY,BC

il registro Indice IY conterrà $4FFF_{16}$.

AND data — AND IMMEDIATO CON L'ACCUMULATORE



AND data

E6 yy

AND del contenuto del successivo byte della memoria di programma con l'Accumulatore.

Supponiamo che xx = $3A_{16}$. Dopo l'esecuzione dell'istruzione

AND 7CH

l'Accumulatore conterrà 38_{16}.

```
3A = 0011  1010
7C = 0111  1100
     ----------
     0011  1000
```

0 posiziona S a 0

Tre bit ad 1, posizionano P/O a 0

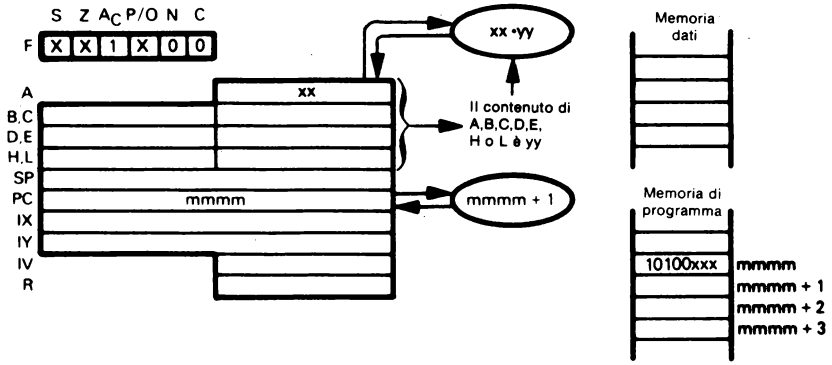
Risultato non zero, posiziona Z a 0

Questa è un'istruzione logica ordinaria; è spesso usata per porre i bit nello stato «off». Per esempio, l'istruzione

AND 7FH

posizionerà incondizionatamente a 0 il bit di ordine maggiore dell'Accumulatore.

AND reg — AND DEL REGISTRO CON L'ACCUMULATORE



AND reg

10100 xxx

000 per reg = B
001 per reg = C
010 per reg = D
011 per reg = E
100 per reg = H
101 per reg = L
111 per reg = A

AND dell'Accumulatore col contenuto del Registro A, B, C, D, E, H o L. Salva il risultato nell'Accumulatore.

Supponiamo che xx = $E3_{16}$ e che il Registro E contenga $A0_{16}$. Dopo l'esecuzione della istruzione

AND E

l'Accumulatore conterrà $A0_{16}$.

```
E3 = 1110  0011
A0 = 1010  0000
     1010  0000
```

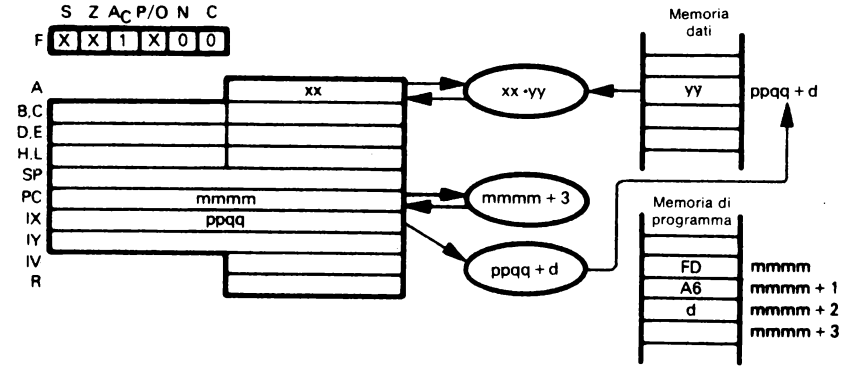
1 posiziona S a 1

Due bit 1, posizionano P/O a 1

Risultato non zero, posiziona Z a 0

AND è una istruzione logica usata frequentemente.

AND (HL) – AND DELLA MEMORIA CON L'ACCUMULATORE
AND (IX + disp)
AND (IY + disp)



L'illustrazione mostra l'esecuzione di AND (IY + disp).

$$\underbrace{\text{AND (IY}}_{\text{FD A6}}\underbrace{\text{+disp)}}_{\text{d}}$$

AND del contenuto della locazione di memoria (specificata dalla somma del contenuto del registro IY e del digit d del dislocamento) con l'Accumulatore.

Supponiamo che xx = $E3_{16}$, ppqq = 4000_{16} e che la locazione di memoria $400F_{16}$ contenga $A0_{16}$. Dopo l'esecuzione dell'istruzione

AND (IY + 0FH)

l'Accumulatore conterrà $A0_{16}$.

```
E3 = 1110 0111
A0 = 1010 0000
     1010 0000
```

1 posiziona S a 1

Due bit ad 1, posizionano P/O a 1

Risultato non zero, posiziona Z a 0

$$\underbrace{\text{AND (IX}}_{\text{DD A6}}\underbrace{\text{+disp)}}_{\text{d}}$$

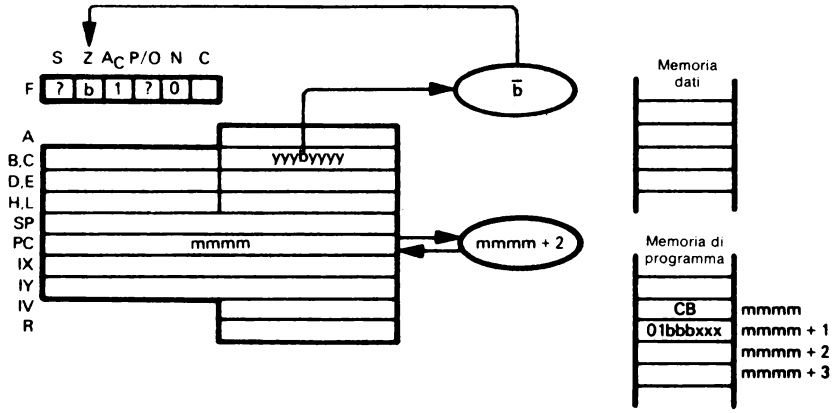
Questa istruzione è identica a AND (IY + disp), tranne che essa usa il registro IX invece del registro IY.

$$\underbrace{\text{AND (HL)}}_{\text{A6}}$$

AND del contenuto della locazione di memoria (specificata dal contenuto della coppia di registri HL) con l'Accumulatore.

AND è un'istruzione logica usata frequentemente.

BIT b,reg — TEST SUL BIT b NEL REGISTRO reg



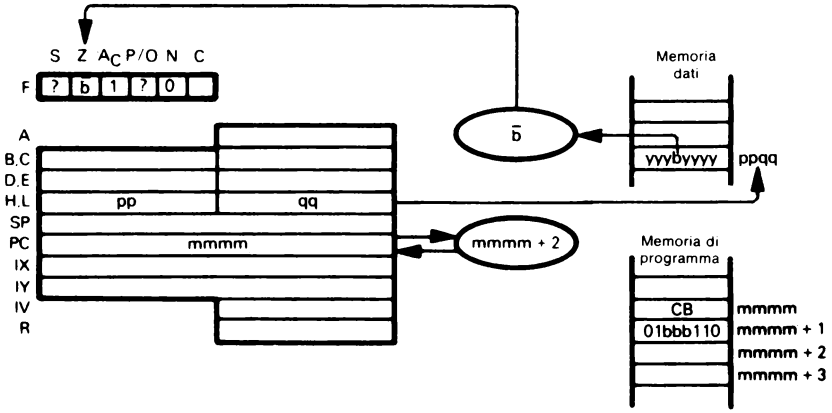
BIT	b,	reg
CB 01	bbb	xxx

Bit provato			Registro
0	000	000	B
1	001	001	C
2	010	010	D
3	011	011	E
4	100	100	H
5	101	101	L
6	110	111	A
7	111		

Pone il complemento del bit specificato del registro indicato nel flag Z del registro F.

Supponiamo che il Registro C contenga 1110 1111. L'istruzione BIT 4,C posizionerà allora ad 1 il flag Z, mentre il bit 4 nel Registro C rimane a 0. Il bit 0 è il bit meno significativo.

BIT b,(HL) — TEST SUL BIT b DELLA POSIZIONE DI MEMORIA
BIT b,(IX + disp) INDICATA
BIT b,(IY + disp)



L'illustrazione mostra l'esecuzione di BIT 4,(HL). Il bit 0 è il bit meno significativo.

BIT	b	(HL)
CB 01	bbb	110

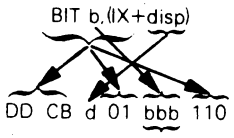
Bit provato	bbb
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Prova il bit indicato nella posizione di memoria specificata dal contenuto del Registro HL e pone il complemento del bit nel flag Z del registro F.

Supponiamo che HL contenga 4000H e che il bit 3 nella locazione di memoria 4000H contenga 1. L'istruzione

BIT 3,(HL)

posizionerà allora a 0 il flag Z, mentre il bit 3 nella locazione di memoria 4000H rimane ad 1.



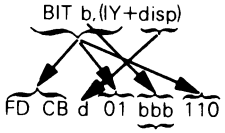
bbb è lo stesso di BIT b, (HL)

Esamina il bit specificato nella locazione di memoria indicata dalla somma del registro Indice IX e di disp. Pone il complemento nel flag Z del registro F.

Supponiamo che il Registro IX contenga 4000H e che il bit 4 della locazione di memoria 4004H sia 0. L'istruzione

BIT 4,(IX + 4H)

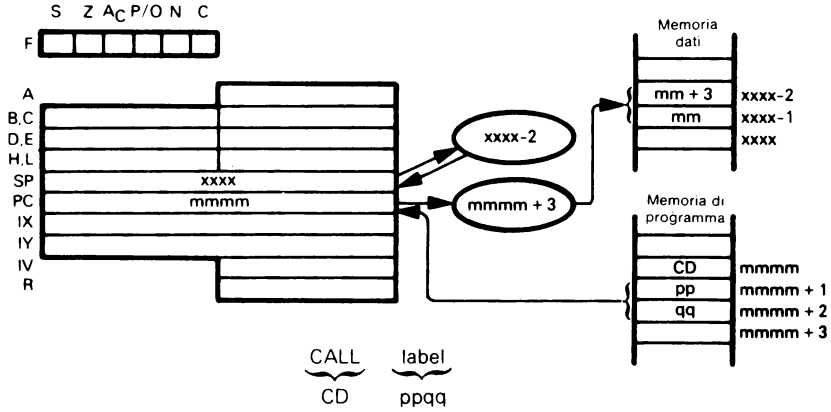
posizionerà allora ad 1 il flag Z, mentre il bit 4 della locazione di memoria 4004H rimane a 0.



bbb è lo stesso di BIT b, (HL)

L'istruzione è identica a BIT b,(IX + disp), tranne che essa usa il registro IY invece del registro IX.

CALL label — CHIAMA IL SOTTOPROGRAMMA IDENTIFICATO NELL'OPERANDO



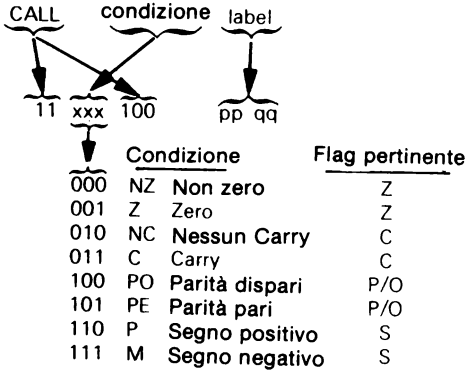
Memorizza l'indirizzo dell'istruzione che segue la CALL in cima allo stack: la cima dello stack è un byte di memoria dati indirizzato dallo Stack Pointer. Si sottrae poi 2 dallo Stack Pointer per indirizzare la nuova cima dello stack. Sposta l'indirizzo a 16 bit contenuto nei byte secondo e terzo del programma oggetto dell'istruzione CALL nel Contatore di Programma. Il secondo byte dell'istruzione CALL è la metà di ordine minore dell'indirizzo e il terzo byte è il byte di ordine maggiore.

Consideriamo la sequenza di istruzioni:

```
        CALL    SUBR
        AND     7CH
        _
        _
        _
SUBR
```

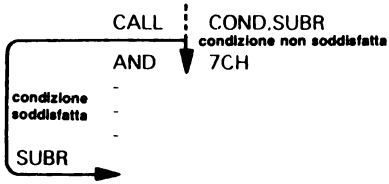
Dopo l'esecuzione dell'istruzione, l'indirizzo dell'istruzione AND è salvato in cima allo stack. Lo Stack Pointer è decrementato di 2. Successivamente si eseguirà l'istruzione SUBR.

CALL condition, label — CHIAMA IL SOTTOPROGRAMMA IDENTIFICATO NELL'OPERANDO SE LA CONDIZIONE È SODDISFATTA



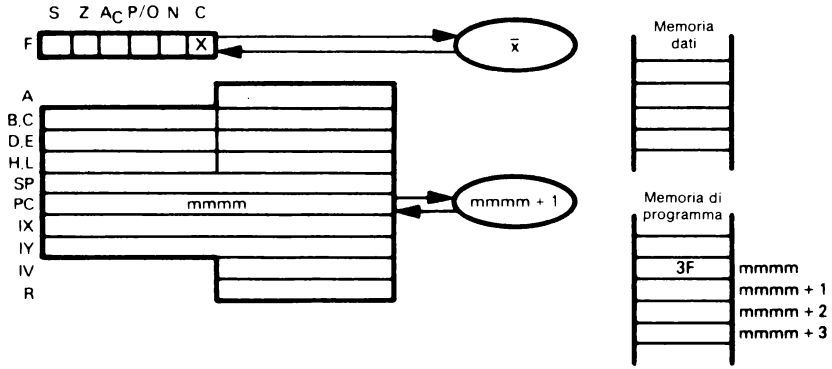
Questa istruzione è identica all'istruzione CALL, tranne che il sottoprogramma identificato sarà chiamato solo se è soddisfatta la condizione; altrimenti sarà eseguita la istruzione che segue in sequenza l'istruzione CALL condition.

Consideriamo la sequenza di istruzioni:



Se la condizione non è soddisfatta, si eseguirà l'istruzione AND dopo l'esecuzione dell'istruzione CALL COND,SUBR. Se la condizione è soddisfatta, l'indirizzo della istruzione AND è salvato in cima allo stack e lo Stack Pointer è decrementato di 2. Successivamente si eseguirà l'istruzione avente l'etichettà SUBR.

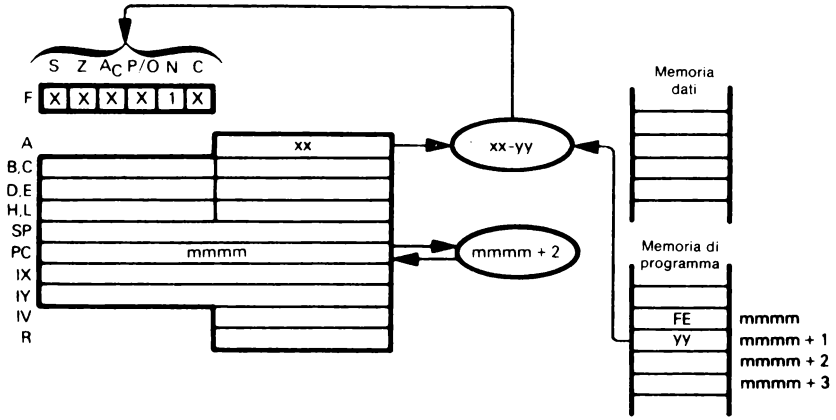
CCF – COMPLEMENTA IL FLAG DI CARRY



CCF
3F

Complementa il flag di Carry. Non si influenza nessun altro stato o contenuto di registri.

CP data – CONFRONTA IMMEDIATAMENTE IL DATO COL CONTENUTO DELL'ACCUMULATORE



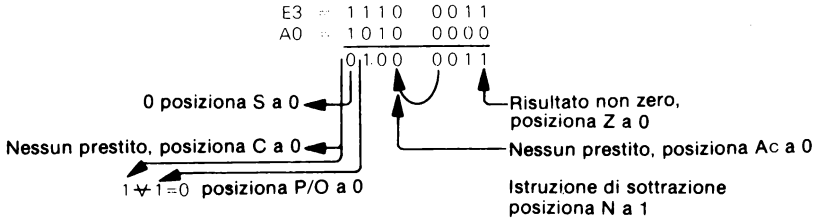
CP data
FE yy

Sottrae il contenuto del secondo byte del codice oggetto dal contenuto dell'Accumulatore, trattando entrambi i numeri come semplici dati binari. Scarta il risultato; cioè, lascia stare l'Accumulatore, ma modifica i flag degli stati per riflettere il risultato della sottrazione.

Supponiamo che xx = $E3_{16}$ e che il secondo byte del codice oggetto dell'istruzione CP contenga $A0_{16}$. Dopo l'esecuzione dell'istruzione

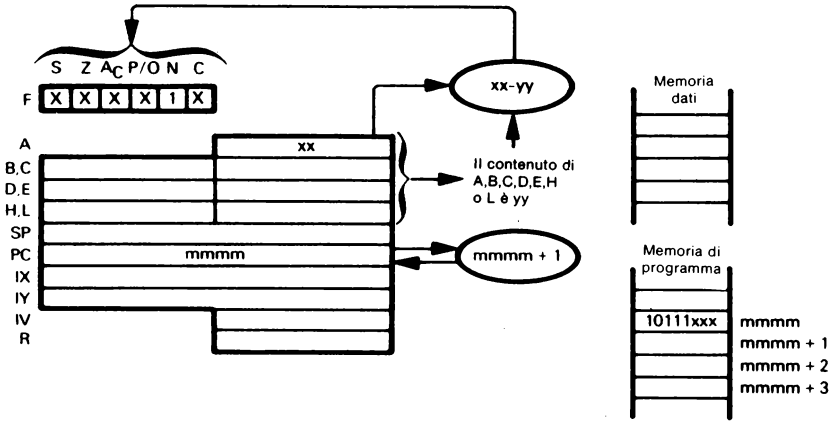
CP 0A0H

l'Accumulatore conterrà ancora $E3_{16}$, ma gli stati saranno modificati come segue:



È da notare che il riporto risultante è complementato.

CP reg — CONFRONTA IL REGISTRO CON L'ACCUMULATORE



CP reg

10111 xxx

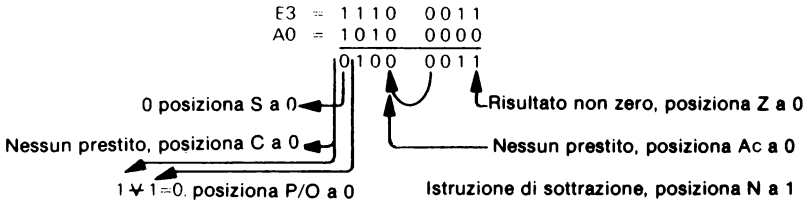
000 per reg = B
001 per reg = C
010 per reg = D
011 per reg = E
100 per reg = H
101 per reg = L
111 per reg = A

Sottrae il contenuto del Registro A, B, C, D, E, H o L dal contenuto dell'Accumulatore, trattando entrambi i numeri come semplici dati binari. Scarta il risultato; cioè lascia stare l'Accumulatore, ma modifica i flag degli stati per riflettere il risultato della sottrazione.

Supponiamo che xx = $E3_{16}$ e che il Registro B contenga $A0_{16}$. Dopo l'esecuzione della istruzione

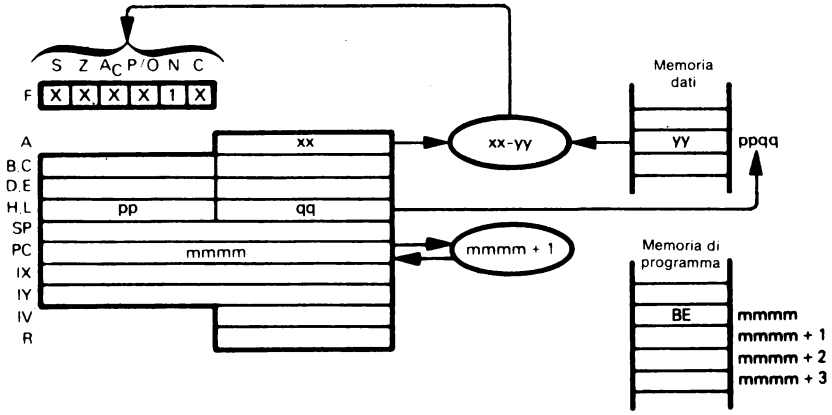
CP B

l'Accumulatore conterrà ancora $E3_{16}$, ma gli stati saranno stati modificati come segue:



È da notare che il carry risultante è complementato.

CP (HL) – CONFRONTA LA MEMORIA CON L'ACCUMULATORE
CP (IX + disp)
CP (IY + disp)



L'illustrazione mostra l'esecuzione di CP (HL):

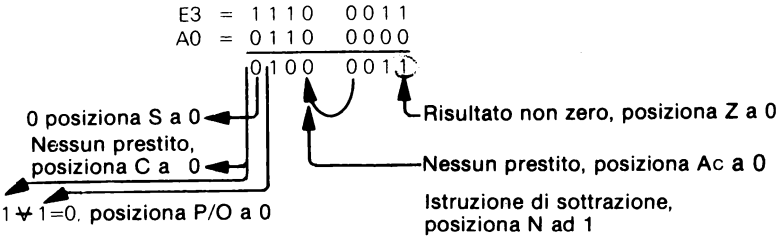
$$\underbrace{\text{CP (HL)}}_{\text{BE}}$$

Sottrae il contenuto della locazione di memoria (specificata dal contenuto della coppia di registri H ed L) dal contenuto dell'Accumulatore, trattando entrambi i numeri come semplici numeri binari. Scarta il risultato; cioè, lascia stare l'Accumulatore, ma modifica i flag degli stati per riflettere il risultato della sottrazione.

Supponiamo che xx = $E3_{16}$ e yy = $A0_{16}$. Dopo l'esecuzione dell'istruzione

CP (HL)

l'Accumulatore conterrà ancora $E3_{16}$, ma gli stati saranno modificati come segue:



È da notare che il carry risultante è complementato.

$$\underbrace{\text{CP (IX}}_{\text{DD BE}}\underbrace{\text{+disp)}}_{\text{d}}$$

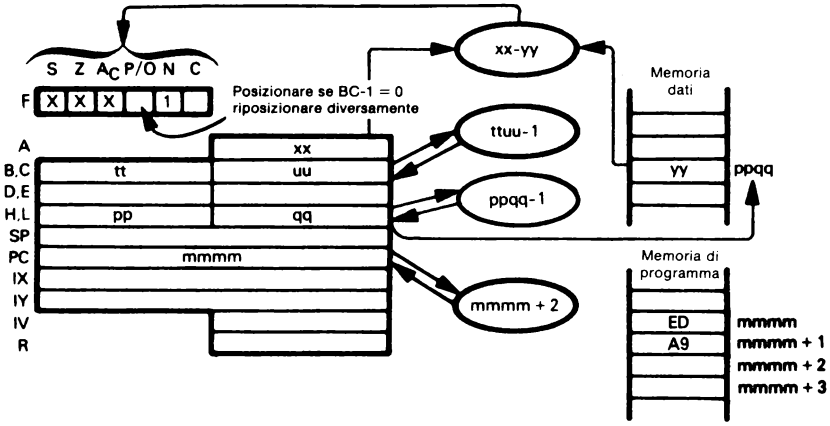
Sottrae il contenuto della locazione di memoria (specificata dalla somma del contenuto del registro IX col valore d del dislocamento) dal contenuto dell'Accumulatore, trattando entrambi i numeri come semplici dati binari. Scarta il risultato; cioè, lascia stare l'Accumulatore, ma modifica i flag degli stati per riflettere il risultato della sottrazione.

CP (IY+disp)

FD BE d

Questa istruzione è identica a CP (IX + disp), tranne che essa usa il registro IY invece del registro IX.

CPD — CONFRONTA L'ACCUMULATORE CON LA MEMORIA. DECREMENTA L'INDIRIZZO E IL CONTATORE DEI BYTE



CPD

ED A9

Confronta il contenuto dell'Accumulatore col contenuto della locazione di memoria (specificata dalla coppia di registri HL). Se A è uguale alla memoria, posiziona il flag Z. Decrementa la coppia di registri HL e BC. (BC è usato come Contatore dei Byte).

Supponiamo che xx = $E3_{16}$, ppqq = 4000_{16}, che BC contenga 0001_{16} e che yy = $A0_{16}$. Dopo l'esecuzione dell'istruzione

CPD

l'Accumulatore conterrà ancora $E3_{16}$, ma gli stati saranno stati modificati come segue:

```
E3 = 1110 0011
A0 = 1010 0000
     0100 0011
```

0 posiziona S a 0

Risultato non zero, posiziona Z a 0

Nessun riporto, posiziona Ac a 0

Il flag P/O sarà azzerato perché BC − 1 = 0

Istruzione di sottrazione interessata, posiziona N a 1

Il carry non è influenzato

La coppia di registri HL conterrà $3FFF_{16}$ e BC = 0.

CPDR — CONFRONTA L'ACCUMULATORE CON LA MEMORIA. DECREMENTA L'INDIRIZZO E IL CONTATORE DEI BYTE. CONTINUA FINCHÈ NON SI TROVA IL BYTE UGUALE O IL CONTATORE DEI BYTE È ZERO

CPDR

ED B9

Questa istruzione è identica a CPD, tranne che essa è ripetuta finchè non si trovi un byte uguale o il contatore dei byte è a zero. Dopo il trasferimento di ogni dato, saranno riconosciute le interruzioni e saranno eseguiti due cicli di rinfresco (refresh).

Supponiamo che la coppia di registri HL contenga 5000_{16}, la coppia di registri BC contenga $00FF_{16}$, l'Accumulatore contenga $F9_{16}$ e che la memoria abbia il seguente contenuto:

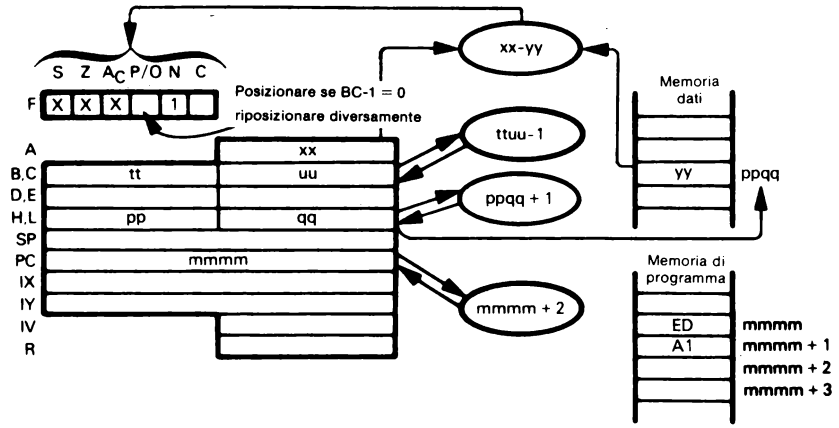
Locazione	Contenuto
5000_{16}	AA_{16}
$4FFF_{16}$	BC_{16}
$4FFE_{16}$	19_{16}
$4FFD_{16}$	$7A_{16}$
$4FFC_{16}$	$F9_{16}$
$4FFB_{16}$	DD_{16}

Dopo l'esecuzione dell'istruzione

CPDR

il flag P/O sarà 1, il flag Z sarà 1, la coppia di registri HL conterrà $4FFB_{16}$ e la coppia di registri BC conterrà $00FA_{16}$.

CPI – CONFRONTA L'ACCUMULATORE CON LA MEMORIA. DECREMENTA IL CONTATORE DEI BYTE. INCREMENTA L'INDIRIZZO



CPI
ED A1

Confronta il contenuto dell'Accumulatore col contenuto della locazione di memoria (specificata dalla coppia di registri HL). Se A è uguale alla memoria, posiziona ad 1 il flag Z. Incrementa la coppia di registri HL e decrementa la coppia di registri BC (BC è usata come Contatore dei Byte).

Supponiamo che xx = $E3_{16}$, ppqq = 4000_{16}, che BC contenga 0032_{16} e che yy = $E3_{16}$. Dopo l'esecuzione dell'istruzione

CPI

l'Accumulatore conterrà ancora $E3_{16}$, ma gli stati saranno modificati come segue:

```
E3  =  1 1 1 1   0 0 1 1
-E3 =  0 0 0 0   1 1 0 1
       ---------------
       0 0 0 0   0 0 0 0
```

0 posiziona S a 0

Il risultato è 0, posiziona Z ad 1

Nessun prestito, posiziona Ac a 0

Il flag P/O sarà posizionato ad 1 perché BC-1 = 0

Istruzione di sottrazione interessata, posiziona N ad 1

Il carry non è influenzato

La coppia di registri HL conterrà 4001_{16} e BC conterrà 0031_{16}.

CPIR — CONFRONTA L'ACCUMULATORE CON LA MEMORIA. DECREMENTA IL CONTATORE DEI BYTE. INCREMENTA L'INDIRIZZO. CONTINUA FINCHÈ NON SI TROVI UN BYTE UGUALE O IL CONTATORE DEI BYTE DIA ZERO

CPIR

ED B1

Questa istruzione è identica a CPI, tranne che essa è ripetuta finché non si trovi un byte uguale o il contatore dei byte sia zero. Dopo ogni trasferimento di un dato, si riconosceranno le interruzioni e si eseguiranno due cicli di rinfresco.

Supponiamo che la coppia di registri HL contenga 4500_{16}, che la coppia di registri BC contenga $00FF_{16}$, e che l'Accumulatore contenga $F9_{16}$ e che la memoria abbia il seguente contenuto:

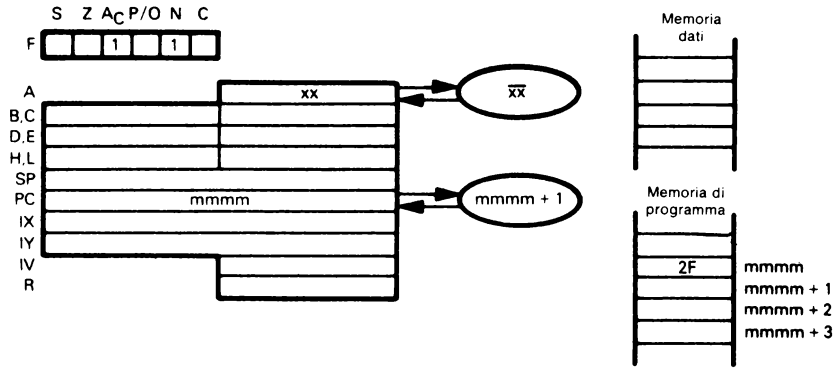
Locazione	Contenuto
4500_{16}	AA_{16}
4501_{16}	15_{16}
4502_{16}	$F9_{16}$

Dopo l'esecuzione di

CPIR

il flag P/O sarà 1 e il flag Z sarà 1. La coppia di registri HL conterrà 4503_{16} e la coppia di registri BC conterrà $00FC_{16}$.

CPL — COMPLEMENTA L'ACCUMULATORE



CPL
2F

Complementa il contenuto dell'Accumulatore. Non si influenza nessun altro contenuto di registri.

Supponiamo che l'Accumulatore contenga $3A_{16}$. Dopo l'esecuzione dell'istruzione

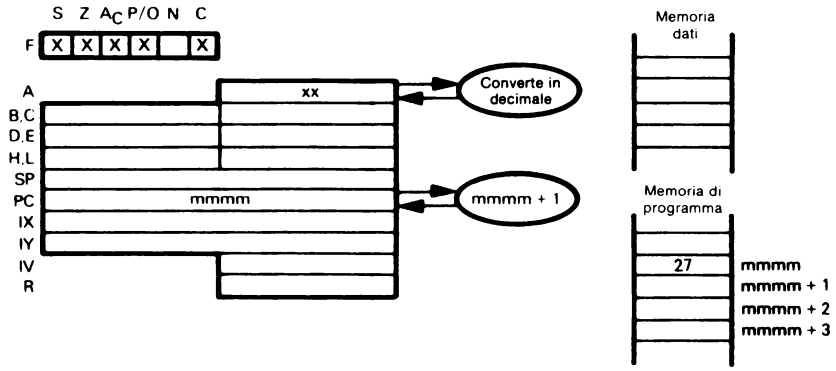
CPL

l'Accumulatore conterrà $C5_{16}$.

3A=0 0 1 1 1 0 1 0
Complemento = 1 1 0 0 0 1 0 1

Questa è una istruzione logica ordinaria. Non bisogna usarla per sottrazione binaria; ci sono speciali istruzioni di sottrazione (SUB, SBC).

DAA — ADATTAMENTO DECIMALE DELL'ACCUMULATORE



DAA
27

Converte il contenuto dell'Accumulatore in una forma binaria con codifica decimale. Questa istruzione potrebbe essere usata solo dopo la somma o la sottrazione di due nueri BCD; cioè, guardate ADD DAA o ADC DAA o INC DAA o SUB DAA o SBC DAA o DEC DAA o NEG DAA come istruzioni aritmetiche decimale composte, che operano su sorgenti BCD per generare risposte BCD.

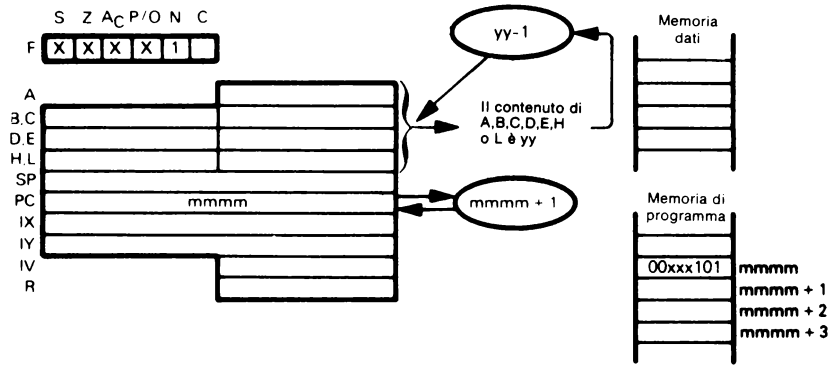
Supponiamo che l'Accumulatore contenga 39_{16} e che il registro B contenga 47_{16}. Dopo l'esecuzione delle istruzioni

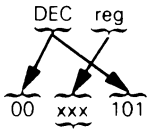
ADD B
DAA

l'Accumulatore conterrà 86_{16}, e non 80_{16}.

La logica della CPU Z80 usa i valori nel Carry e nel Ausiliary Carry, come pure il contenuto dell'Accumulatore, nell'operazione di Adattamento Decimale (Decimal Adjust).

DEC reg – DECREMENTA IL CONTENUTO DEL REGISTRO





000 **per reg** =B
001 **per reg** =C
010 **per reg** =D
011 **per reg** =E
100 **per reg** =H
101 **per reg** =L
111 **per reg** =A

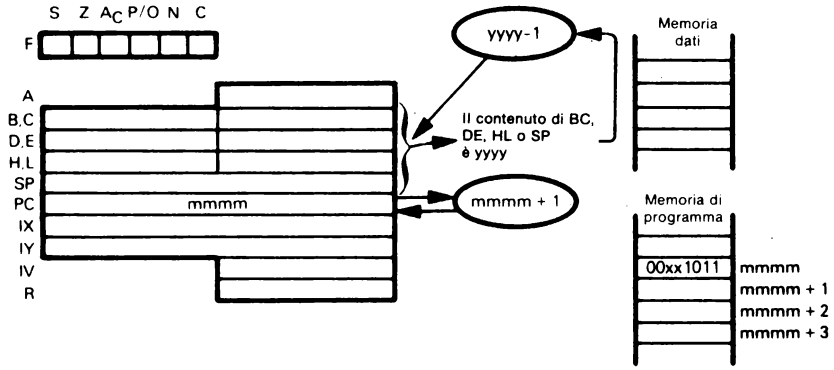
Sottrae 1 dal contenuto del registro specificato.

Supponiamo che il Registro A contenga 50_{16}. Dopo l'esecuzione di

DEC A

il Registro A conterrà $4F_{16}$.

DEC rp — DECREMENTA IL CONTENUTO DELLA COPPIA
DEC IX DI REGISTRI SPECIFICATA
DEC IY



L'illustrazione mostra l'esecuzione di DEC rp:

DEC rp

00 xx 1011

00 per rp è la coppia di registri BC
01 per rp è la coppia di registri DE
10 per rp è la coppia di registri HL
11 per rp è lo Stack Pointer

Sottrae 1 dal valore a 16 bit contenuto nella coppia di registri specificata. Non si influenza nessun flag di stato.

Supponiamo che i registri H ed L contengano $2F00_{16}$; dopo l'esecuzione dell'istruzione

DEC HL

i registri H ed L conterranno $2EFF_{16}$.

DEC IY

FD 2B

Sottrae 1 dal valore a 16 bit contenuto nel registro IX.

DEC IX

DD 2B

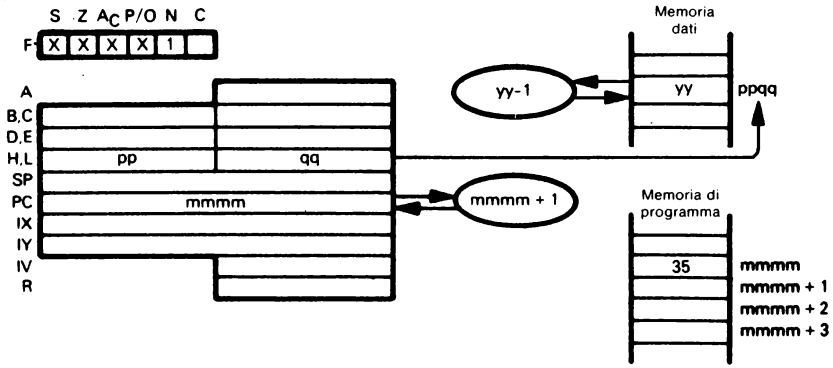
Sottrae 1 dal valore a 16 bit contenuto nel registro IY.

Nè DEC rp, nè DEC IX, nè DEC IY influenzano i flag di stato. Questo è un difetto dell'insieme delle istruzioni dello Z80, ereditato dallo 8080.Mentre si usa l'istruzione DEC reg in loop di istruzioni iterative che usano un contatore con valore 256 o minore, l'istruzione DEC rp (DEC IX o DEX IY) deve essere usata se il valore del contatore è maggiore di 256. Poichè l'istruzione

DE rp non posiziona nessun flag, si devono aggiungere altre istruzioni semplicemente per verificare se il risultato è zero. Ecco una forma tipica di loop:

```
        LD      DE,DATA ;CARICA IL VALORE INIZIALE DEL CONTATORE A 16 BIT
LOOP    —               ;PRIMA ISTRUZIONE DEL LOOP
        —
        —
        DEC     DE      ;DECREMENTA IL CONTATORE
        LD      A,D     ;PER VERIFICARE SE È ZERO, SPOSTA D IN A
        OR      E       ;QUINDI FA L'OR DI A CON E
        JP      NZ,LOOP ;RITORNA SE NON È ZERO
```

DEC (HL) — DECREMENTA IL CONTENUTO DELLA MEMORIA
DEC (IX + disp)
DEC (IY + disp)



L'illustrazione mostra l'esecuzione di DEC (HL):

$$\underbrace{\text{DEC (HL)}}_{35}$$

Sottrae 1 dal contenuto della locazione di memoria (specificata dal contenuto della coppia di registri HL).

Supponiamo che ppqq = 4500_{16}, yy = $5F_{16}$. Dopo l'esecuzione di

DEC (HL)

la locazione di memoria 4500_{16} conterrà $5E_{16}$.

```
 5F  =  0 1 0 1   1 1 1 1
-01  =  1 1 1 1   1 1 1 1
        0 1 0 1   1 1 1 0
```

0 posiziona S a 0

1 ⊻ 1=0, posiziona P/O a 0

Risultato non zero, posiziona Z a 0

Nessun riporto, posiziona Ac a 0

Istruzione di sottrazione, posiziona N ad 1

DEC (IX+disp)

DD 35 d

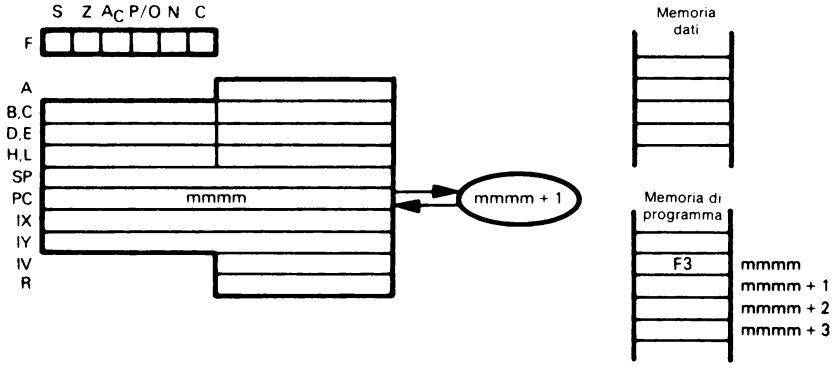
Sottrae 1 dal contenuto della locazione di memoria (specificata dalla somma del contenuto del registro IX col valore d del dislocamento).

DEC (IY+disp)

FD 35 d

Questa istruzione è identica a DEC (IX + disp), tranne che essa usa il registro IY invece del registro IX.

DI — DISABILITA LE INTERRUZIONI



DI

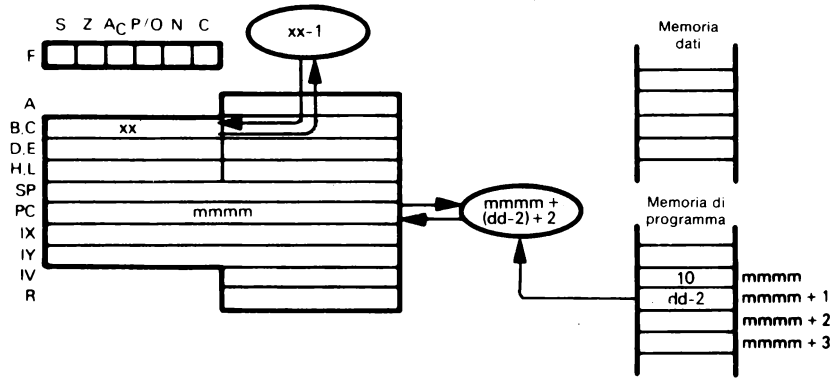
F3

Quando si esegue questa istruzione, si disabilita la richiesta delle interruzioni mascherabili e si ignorerà l'ingresso $\overline{INT}$ sulla CPU. È da ricordare che quando un'interruzione viene riconosciuta, si disabilita automaticamente l'interruzione mascherabile.

La richiesta dell'interruzione mascherabile rimane disabilitata finchè essa non venga successivamente abilitata da un'istruzione EI.

Nessun registro o flag è influenzato da questa istruzione.

DJNZ disp — SALTO RELATIVO AL CONTENUTO PRESENTE DEL CONTATORE DEI PROGRAMMI SE IL REG B NON È ZERO



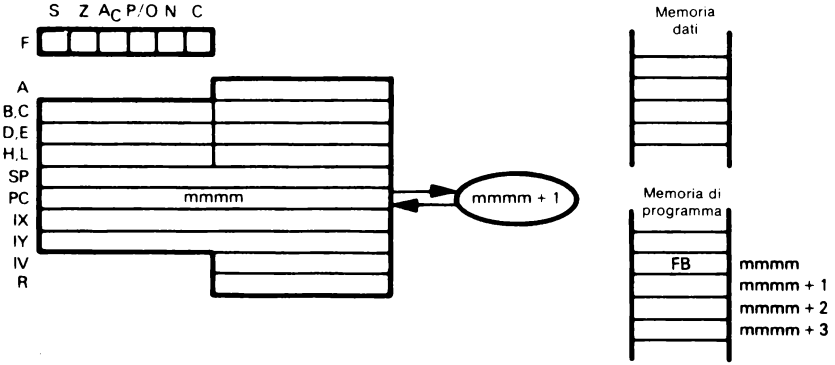
DJNZ disp
10 dd-2

Decrementa il Registro B. Se il contenuto che rimane non è zero, somma il contenuto del secondo byte del codice oggetto dell'istruzione DJNZ più 2 al Contatore di Programma. Il salto è misurato dall'indirizzo del codice operativo dell'istruzione, ed ha un campo compreso tra —126 e +129 byte. L'Assembler automaticamente si regola per l'incremento doppio del PC.

Se il contenuto di B è zero dopo il decremento, si esegue la successiva istruzione in sequenza.

L'istruzione DJNZ è estremamente utile per ogni operazione in un loop di un programma, poichè una sola istruzione sostituisce la tipica sequenza d'istruzioni «decrementa quindi salta su condizione».

EI — ABILITA LE INTERRUZIONI



EI
FB

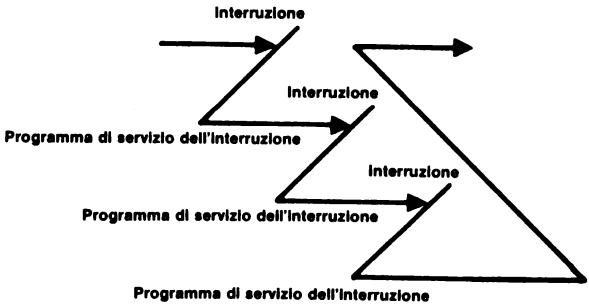
L'esecuzione di questa istruzione provoca la disabilitazione delle interruzioni, ma non finchè non si esegue almeno un'istruzione.

La maggior parte dei programmi di servizio delle interruzioni finisce con le due istruzioni:

```
EI          ;ABILITA LE INTERRUZIONI
RET         ;RITORNA AL PROGRAMMA INTERROTTO
```

Se le interruzioni subiscono un processo in serie, allora tutte le interruzioni mascherabili sono disabilitate per l'intera durata del programma di servizio delle interruzioni – che significa che in un'applicazione a multi-interruzione c'è una significativa possibilità che siano sospese una o più interruzioni quando un programma di servizio di un'interruzione completa la sua esecuzione.

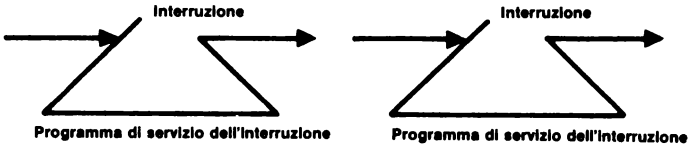
Se le interruzioni fossero riconosciute non appena si è eseguita l'istruzione EI, allora non si eseguirebbe l'istruzione Return. In questo caso, i ritorni si accumulerebbero uno sopra l'altro – e necessariamente consumerebbero spazio della memoria della catasta (stack). Ciò può essere illustrato come segue:



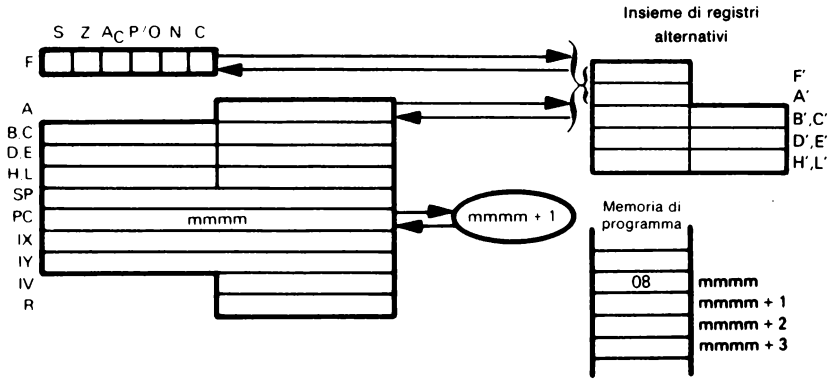
Inibendo le interruzioni per almeno un'istruzione seguente l'esecuzione di EI, la CPU Z80 assicura che l'istruzione RET sarà eseguita in sequenza:

```
–
–
–
EI          ;ABILITA LE INTERRUZIONI
RET         ;RITORNO DALL'INTERRUZIONE
```

Non è raro che le interruzioni siano tenute disabilitate durante l'esecuzione di un programma di servizio di un'interruzione. Le interruzioni subiscono un processo serialmente:



EX AF,AF' — SCAMBIA LO STATO DEL PROGRAMMA E LO STATO DEL PROGRAMMA ALTERNATIVO



EX AF,AF'
08

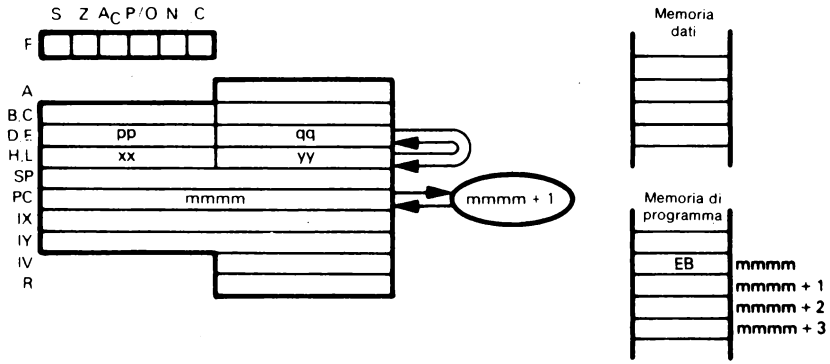
Il contenuto dei due byte delle coppie di registri AF e A'F' vengono scambiati.

Supponiamo che AF contenga $4F99_{16}$ e che A'F' contenga $10AA_{16}$. Dopo l'esecuzione di

EX AF,AF'

AF conterrà $10AA_{16}$ e AF' conterrà $4F99_{16}$.

EX DE,HL — SCAMBIA I CONTENUTI DI DE ED HL



EX DE,HL
EB

Il contenuto dei registri D ed E è scambiato col contenuto dei registri H ed L.

Supponiamo che pp = 03_{16}, qq = $2A_{16}$, xx = 41_{16} e che yy = FC_{16}. Dopo l'esecuzione dell'istruzione

```
EX DE,HL
```

H conterrà 03_{16}, L conterrà $2A_{16}$, D conterrà 41_{16} ed E conterrà FC_{16}.

Le due istruzioni:

```
EX DE,HL
LD A,(HL)
```

sono equivalenti a:

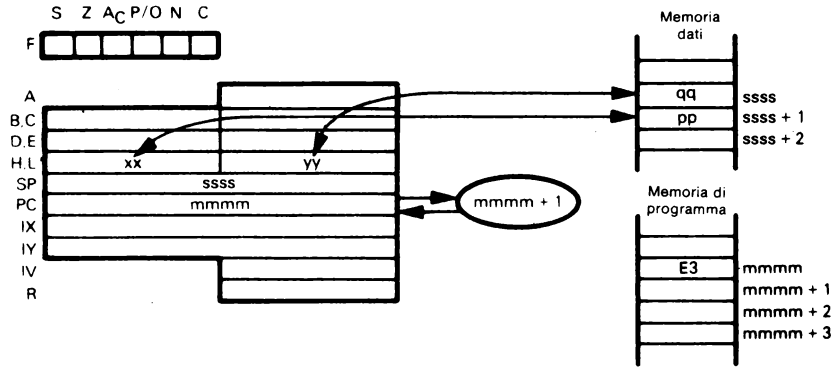
```
LD A,(DE)
```

ma se volete caricare nel registro B il dato indirizzato dai registri d ed E,

```
EX DE,HL
LD B(HL)
```

non hanno nessuna singola istruzione equivalente.

EX (SP),HL – SCAMBIA IL CONTENUTO DEL REGISTRO
EX (SP),IX E DELLA SOMMITÀ DELLO STACK
EX (SP),IY



L'illustrazione mostra l'esecuzione di EX (SP),HL.

$$\underbrace{\text{EX (SP),HL}}_{\text{E3}}$$

Scambia il contenuto del registro L con il byte in cima allo stack. Scambia il contenuto del registro H col byte che sta sotto alla cima dello stack.

Supponiamo che xx = 21_{16}, yy = FA_{16}, pp = $3A_{16}$, qq = $E2_{16}$. Dopo l'esecuzione della istruzione

EX (SP),HL

H conterrà $3A_{16}$, L conterrà $E2_{16}$ e i due byte sulla sommità dello stack conterranno rispettivamente FA_{16} e 21_{16}.

L'istruzione EX (SP),HL è usata per accedere e manipolare i dati sulla sommità dello stack.

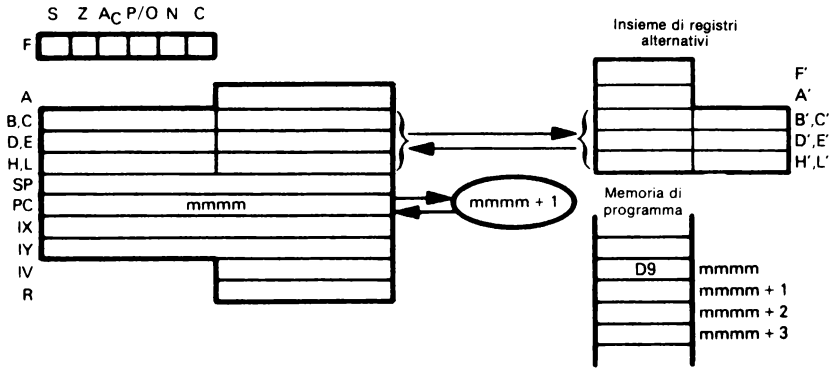
$$\underbrace{\text{EX (SP),IX}}_{\text{DD E3}}$$

Scambia il contenuto del byte di ordine minore del registro IX col byte in cima allo stack. Scambia il byte di ordine maggiore del registro IX col byte che sta sotto alla sommità dello stack.

$$\underbrace{\text{EX (SP),IY}}_{\text{FD E3}}$$

Questa istruzione è identica a EX (SP),IX, ma usa il registro IY invece di IX.

EXX — SCAMBIA LA COPPIA DI REGISTRI CON LA COPPIA DI REGISTRI ALTERNATIVA



EXX
D9

Il contenuto delle coppie di registri BC, DE e HL è scambiato col contenuto delle coppie di registri B'C', D'E' e H'L'.

Supponiamo che le coppie di registri BC, DE e HL contengano rispettivamente 4901_{16}, $5F00_{16}$ e 7251_{16} e che le coppie di registri B'C', D'E' e H'L' contengano rispettivamente 0000_{16}, $10FF_{16}$ e 3333_{16}. Dopo l'esecuzione di

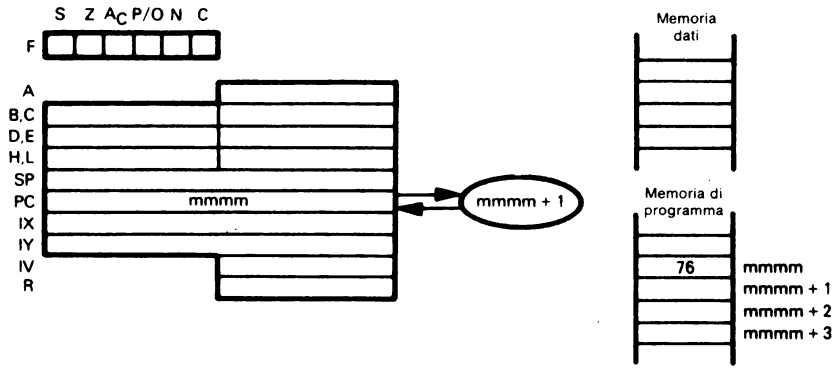
EXX

i registri avranno i seguenti contenuti:

BC: 0000_{16}; DE: $10FF_{16}$; HL: 3333_{16};
B'C': 4901_{16}; D'E': $5F00_{16}$;H'L': 7251_{16}

Questa istruzione può essere usata per scambiare banchi di registri per ottenere tempi di risposta alle interruzioni molto veloci.

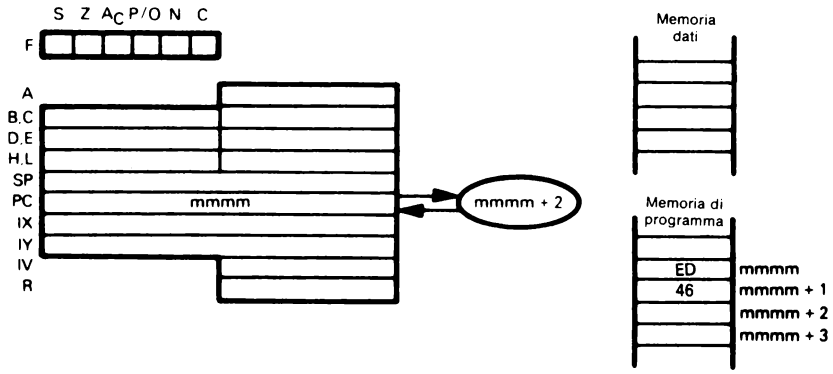
HALT



HALT
76

Quando si esegue l'istruzione HALT, l'esecuzione del programma cessa. La CPU richiede un'interruzione o un reset per far ripartire l'esecuzione. Non vengono influenzati nè i registri nè gli stati; tuttavia, la logica di rinfresco della memoria continua a funzionare.

IM 0 — INTERRUZIONE DI MODO 0



IM 0
ED 46

Questa istruzione pone la CPU nel modo 0 d'interruzione. In questo modo, il dispositivo interrompente metterà un'istruzione sul Bus dei Dati e la CPU eseguirà quella istruzione. Non si influenza nessun registro o stato.

IM 1 — INTERRUZIONE DI MODO 1

IM 1
ED 56

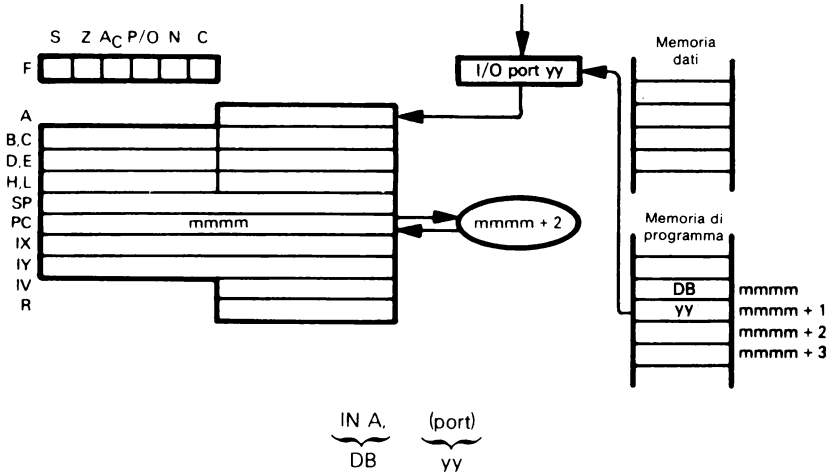
Questa istruzione mette la CPU nel modo 1 d'interruzione. In questo modo, la CPU risponde ad una interruzione eseguendo un restart (RST) alla locazione 0038_{16}.

IM 2 — INTERRUZIONE DI MODO 2

IM 2
ED 5E

Questa istruzione mette la CPU nel modo 2 d'interruzione. In questo modo, la CPU effettua una chiamata indiretta ad una locazione specificata in memoria. Si forma un indirizzo a 16 bit usando il contenuto del registro del Vettore d'Interruzione (IV) per gli otto bit superiori, mentre gli otto bit minori sono forniti dal dispositivo che interrompe. Ci si riferisca al Capitolo 5 per una descrizione completa dei modi di interruzione. Questa istruzione non influenza nessun registro o stato.

IN A,(port) — INGRESSO NELL'ACCUMULATORE



Carica un byte dati nell'Accumulatore dalla porta di I/O (identificata dal secondo byte del codice oggetto dell'istruzione IN).

Supponiamo che 36_{16} sia contenuto nel buffer della porta di I/O $1A_{16}$. Dopo l'esecuzione dell'istruzione

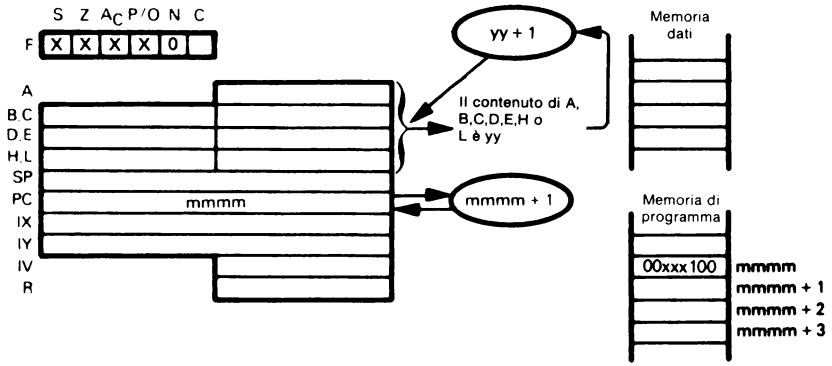
IN A,(1AH)

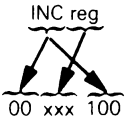
l'Accumulatore conterrà 36_{16}.

L'istruzione IN non influenza nessuno stato.

L'uso dell'istruzione IN è molto dipendente dall'hardware. Gli indirizzi validi per le porte di I/O sono determinati dal modo di implementazione della logica di I/O. È pure possibile progettare un sistema a microcalcolatore che accede alla logica esterna usando istruzioni di riferimento alla memoria con specifici indirizzi di memoria.

INC reg — INCREMENTA IL CONTENUTO DEL REGISTRO





000 **per** reg=B
001 **per** reg=C
010 **per** reg=D
011 **per** reg=E
100 **per** reg=H
101 **per** reg=L
111 **per** reg=A

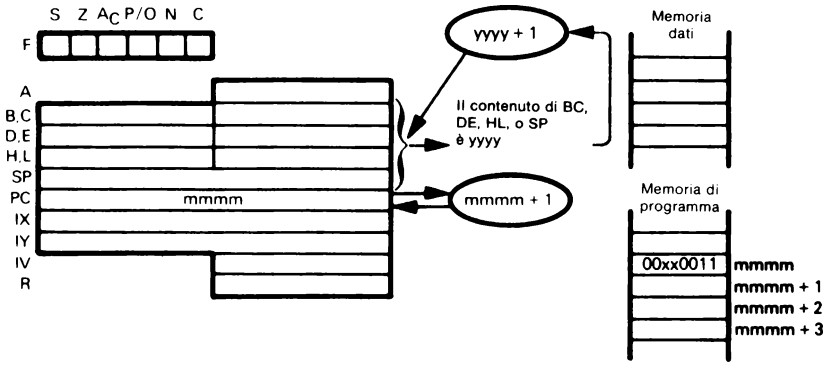
Somma 1 al contenuto del registro specificato.

Supponiamo che il Registro E contenga $A8_{16}$. Dopo l'esecuzione di

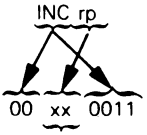
INC E

il registro E conterrà $A9_{16}$.

INC rp – INCREMENTA IL CONTENUTO DELLA COPPIA
INC IX DI REGISTRI SPECIFICATA
INC IY



L'illutrazione mostra l'esecuzione di INC rp:



00 per rp è la coppia di registri BC
01 per rp è la coppia di registri DE
10 per rp è la coppia di registri HL
11 per rp è lo Stack Pointer

Somma 1 al valore di 16 bit contenuto nella coppia di registri specificata. Non sono influenzati i flag di stato.

Supponiamo che i registri D ed E contengano $2F7A_{16}$. Dopo l'esecuzione dell'istruzione

INC DE

i registri D ed E conterranno $2F7B_{16}$.

INC IX
DD 23

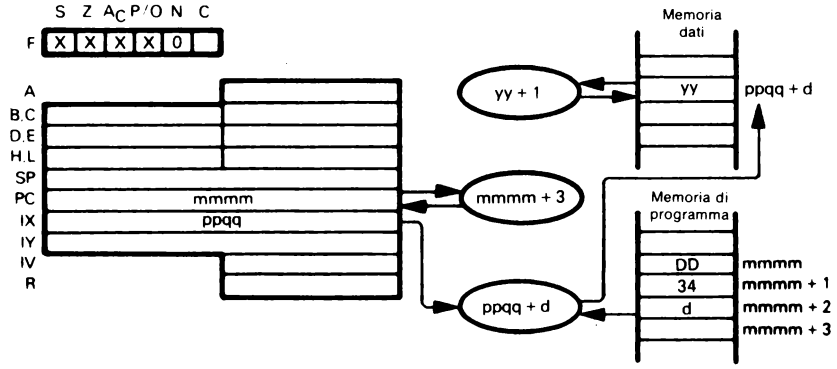
Somma 1 al valore di 16 bit contenuto nel registro IX.

INC IY
FD 23

Somma 1 al valore di 16 bit contenuto nel registro IY.

Così come DEC rp, DEC IX e DEC IY anche INC rp, INC IX e INC IY non influenzano nessun flag di stato. Questo è un difetto dell'insieme d'istruzioni dello Z80 ereditato dallo 8080.

INC (HL) — INCREMENTA IL CONTENUTO DELLA MEMORIA
INC (IX + disp)
INC (IY + disp)



L'illustrazione mostra l'esecuzione di INC (IX + d):

$$\underbrace{\text{INC (IX}}\underbrace{\text{+disp)}}$$

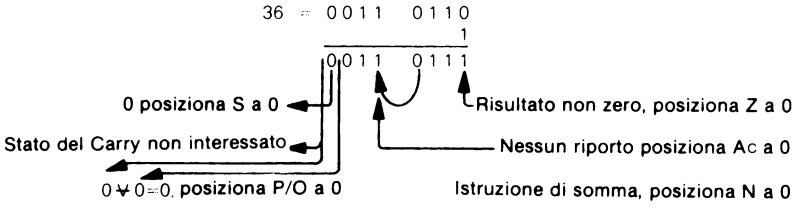
DD 34 d

Somma 1 al contenuto della locazione di memoria (specificata dalla somma del contenuto del Registro IX e del valore d del dislocamento).

Supponiamo che ppqq = 4000_{16} e che la locazione di memoria $400F_{16}$ contenga 36_{16}. Dopo l'esecuzione dell'istruzione

INC (IX + 0FH)

la locazione di memoria $400F_{16}$ conterrà 37_{16}.



$$\underbrace{\text{INC (IY}}\underbrace{\text{+disp)}}$$

FD 34 d

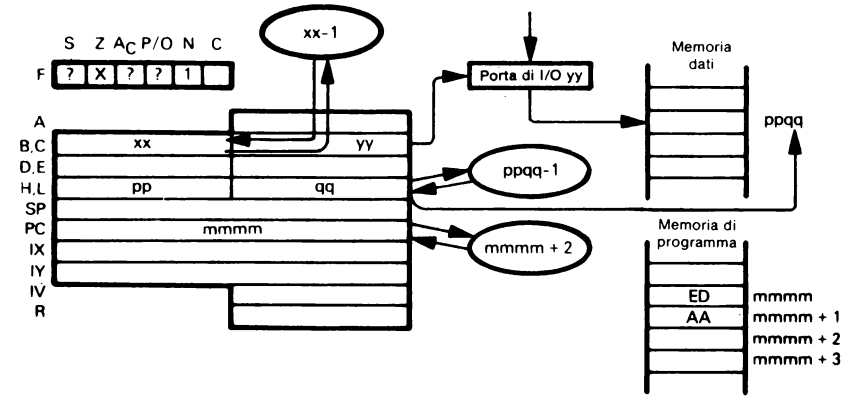
Questa istruzione è identica a INC (IX + disp), tranne che essa usa il registro IY invece che il registro IX.

$$\underbrace{\text{INC (HL)}}$$

34

Somma 1 al contenuto della locazione di memoria (specificata dal contenuto della coppia di registri HL).

IND — INGRESSO IN MEMORIA A DECREMENTO DEL PUNTATORE



IND
ED AA

Ingresso di una porta di I/O (indirizzata dal Registro C) in una locazione di memoria (specificata da HL). Decremento dei Registri B e HL.

Supponiamo che xx = 05_{16}, yy = 15_{16}, ppqq = 2400_{16} e che nel buffer della porta di I/O 15_{16} ci sia 19_{16}. Dopo l'esecuzione dell'istruzione

IND

la locazione di memoria 2400_{16} conterrà 19_{16}. Il registro B conterrà 04_{16} e la coppia di registri HL conterrà $23FF_{16}$.

INDR — INGRESSO IN MEMORIA E DECREMENTO DEL PUNTATORE FINCHÈ IL CONTATORE DEI BYTE NON SIA ZERO

INDR
ED BA

INDR è identica a IND, ma si ripete finchè il Registro B = 0.

Supponiamo che il Registro B contenga 03_{16}, che il Registro C contenga 15_{16} e che HL contenga 2400_{16}. La seguente sequenza di byte sia disponibile alla porta di I/O 15_{16}:

17_{16}, 59_{16}, e AE_{16}

Dopo l'esecuzione di

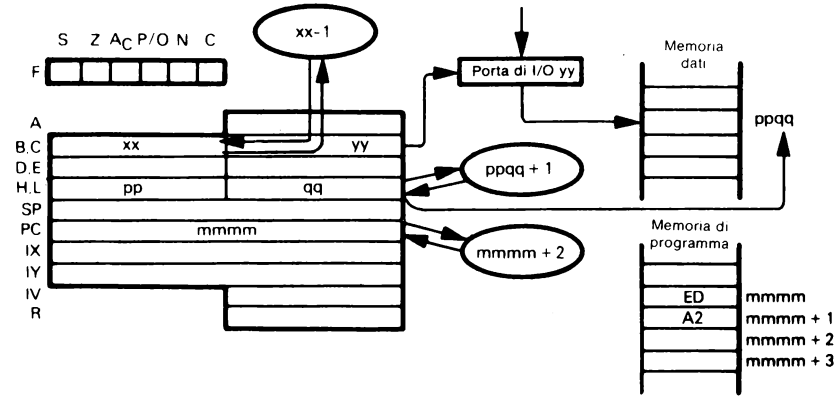
INDR

la coppia di registri HL conterrà $23FD_{16}$ e il Registro B conterrà zero e le locazioni di memoria avranno i contenuti seguenti:

Locazione	Contenuto
2400	17_{16}
23FF	59_{16}
23FE	AE_{16}

Questa istruzione è estremamente utile per caricare blocchi di dati da un dispositivo di ingresso in memoria.

INI — INGRESSO IN MEMORIA E INCREMENTO DEL PUNTATORE



INI
ED A2

Ingresso da una porta di I/O (indirizzata dal Registro C) in una locazione di memoria (specificata da HL). Decremento del Registro B; incremento della coppia di registri HL.

Supponiamo che xx = 05_{16}, yy = 15_{16}, ppqq = 2400_{16} e che nel buffer della porta di I/O 15_{16} ci sia 19_{16}. Dopo l'esecuzione dell'istruzione

INI

la locazione di memoria 2400_{16} conterrà 19_{16}. Il registro B conterrà 04_{16} e la coppia di registri HL conterrà 2401_{16}.

INIR — INGRESSO IN MEMORIA E DECREMENTO DEL PUNTATORE FINCHÈ IL CONTATORE DEI BYTE NON SIA ZERO

INIR
ED B2

INIR è identico a INI, ma si ripete finchè il Registro B = 0.

Supponiamo che il Registro B contenga 03_{16}, che il Registro C contenga 15_{16} e che HL contenga 2400_{16}. La seguente sequenza di byte sia disponibile alla porta di I/O 15_{16}.

17_{16}, 59_{16} e AE_{16}

Dopo l'esecuzione di

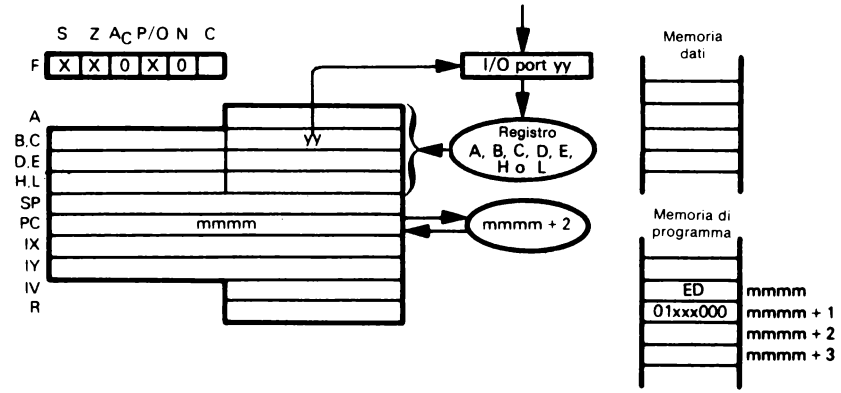
INIR

la coppia di registri HL conterrà 2403_{16} ed il Registro B conterrà zero e le locazioni di memoria avranno i contenuti seguenti:

Locazione	Contenuto
2400	17_{16}
2401	59_{16}
2402	AE_{16}

Questa istruzione è estremamente utile per caricare blocchi di dati da un dispositivo in memoria.

IN reg,(C) – INGRESSO IN UN REGISTRO



IN reg, (C)

ED 01 xxx 000

000 per reg=B
001 per reg=C
010 per reg=D
011 per reg=E
100 per reg=H
101 per reg=L
111 per reg=A
110 per posizionare i flag di stato
senza cambiare registri

Carica un byte di dato nel registro specificato (reg) dalla porta di I/O (identificata dal contenuto del registro C).

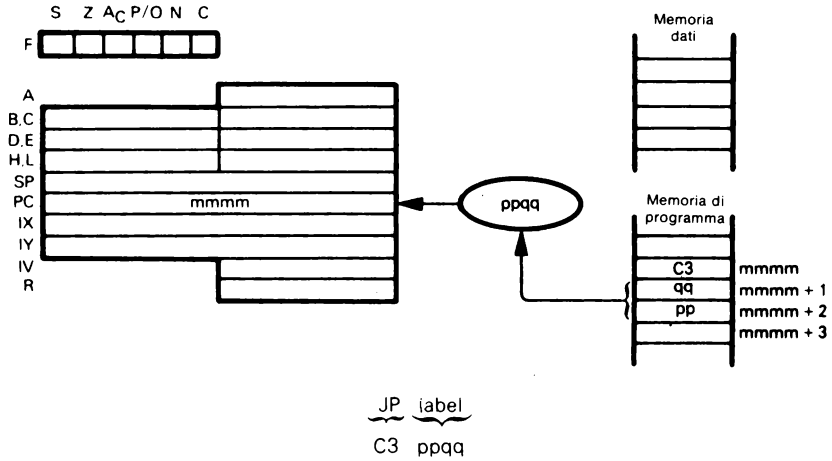
Supponiamo che 42_{16} sia contenuto nel buffer della porta di I/O 36_{16} e che il Registro C contenga 36_{16}. Dopo l'esecuzione dell'istruzione

IN D,(C)

il registro D conterrà 42_{16}.

Durante l'esecuzione dell'istruzione, il contenuto del Registro B è posto sulla metà superiore del Bus degli Indirizzi, rendendolo capace di estendere il numero delle porte di I/O indirizzabili.

JP label — SALTA ALL'ISTRUZIONE IDENTIFICATA NELL'OPERANDO



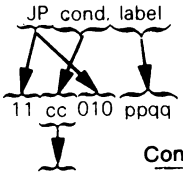
Carica il contenuto del secondo e del terzo byte del codice oggetto dell'istruzione JP nel Contatore dei Programmi; questo diventa l'indirizzo di memoria della prossima istruzione da eseguire. Il contenuto precedente del Contatore dei Programmi è perduto.

Nella sequenza seguente:

```
        JP      NEXT
        HAND    7FH
        —
        —
NEXT    CPL
```

L'istruzione CPL sarà eseguita dopo l'esecuzione dell'istruzione JP. L'istruzione AND non sarà mai eseguita, a meno che in qualche altro posto della sequenza delle istruzioni una istruzione JP non salti a questa istruzione.

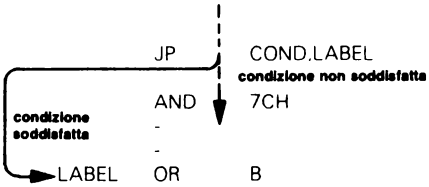
JP condition, label – SALTA ALL'INDIRIZZO IDENTIFICATO NELL'OPERANDO SE LA CONDIZIONE È SODDISFATTA



		Condizione	Flag pertinente
000	NZ	Non-Zero	Z
001	Z	Zero	Z
010	NC	**Nessun Carry**	C
011	C	Carry	C
100	PO	**Parità dispari**	P/O
101	PE	**Parità pari**	P/O
110	P	**Segno positivo**	S
111	M	**Segno negativo**	S

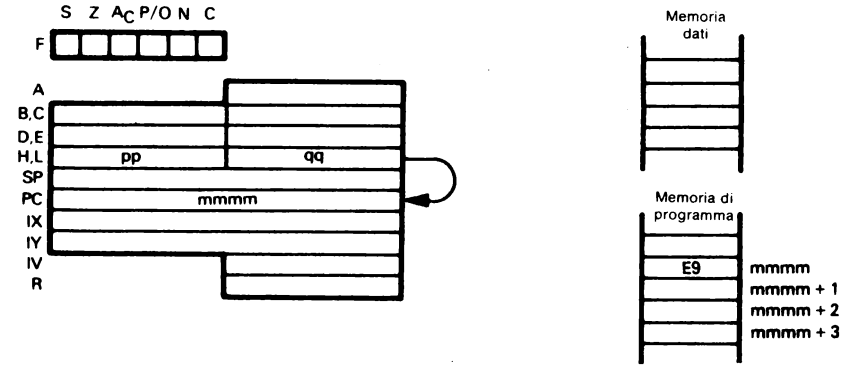
Questa istruzione è identica all'istruzione IP, tranne che il salto sarà effettuato solo se la condizione è soddisfatta; altrimenti, si eseguirà l'istruzione seguente in sequenza l'istruzione JP condition.

Consideriamo la sequenza di istruzioni:



Dopo l'esecuzione dell'istruzione JP cond,label, se la condizione è soddisfatta si eseguirà allora l'istruzione OR. Se la condizione non è soddisfatta, si eseguirà la istruzione AND, che è l'istruzione successiva della sequenza.

JP (HL) – SALTO ALL'INDIRIZZO SPECIFICATO DAL CONTENUTO
JP (IX) DEL REGISTRO A 16 BIT
IP (IY)



L'illustrazione mostra l'esecuzione di JP (HL):

$$\underbrace{\text{JP (HL)}}_{\text{E9}}$$

Il contenuto della coppia di registri HL è messo nel Contatore di Programma: si realizza, quindi, un salto con indirizzamento implicito.

La sequenza d'istruzioni

```
LD      H,ADDR
JP      (HL)
```

ha esattamente lo stesso effetto netto dell'istruzione singola

```
JP      ADDR
```

Entrambe specificano che successivamente si deve eseguire l'istruzione avente come label ADDR.

L'istruzione JP (HL) è utile quando si vuole incrementare un indirizzo di ritorno per un sottoprogramma che ha ritorni multipli.

Consideriamo la seguente chiamata al sottoprogramma SUB:

```
CALL    SUB     ;CHIAMATA DEL SOTTOPROGRAMMA
JP      ERR     ;RITORNO ERRORE
                ;RITORNO BUONO
```

Usando RET per tornare da SUB si tornerebbe all'esecuzione di JP ERR; perciò, se si esegue SUB senza condizioni che rivelino errori, si ritorna come segue:

```
POP     HL      ;ESTRAE L'INDIRIZZO DI RITORNO PER HL
INC     HL      ;SOMMA TRE ALL'INDIRIZZO DI RITORNO
INC     HL
INC     HL
IP      (HL)    ;RITORNO
```

JP (IX)
DD E9

Questa istruzione è identica all'istruzione JP (HL), tranne che essa usa il registro IX invece della coppia di registri HL.

JP (IY)
FD E9

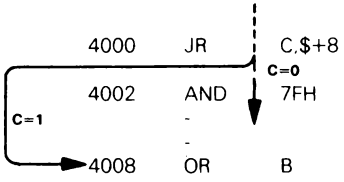
Questa istruzione è identica all'istruzione JP (HL), tranne che essa usa il registro IY invece della coppia di registri HL.

JR C,disp — SALTO RELATIVO AL CONTENUTO DEL CONTATORE DI PROGRAMMA SE IL CARRY È POSTO AD 1

JR C, disp
38 dd-2

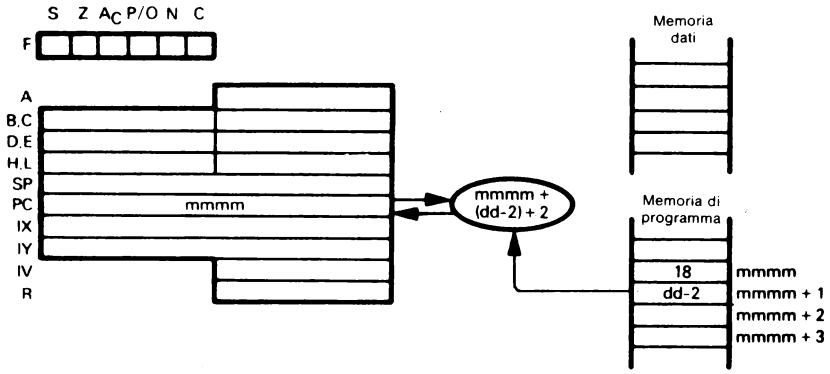
Questa istruzione è identica all'istruzione JR disp, tranne che si esegue il salto solo se lo stato di Carry è uguale a 1; altrimenti, si esegue la istruzione successiva.

Nella seguente sequenza di istruzioni:



Dopo l'istruzione JR C,$ + 8, si esegue l'istruzione OR se lo stato di Carry è uguale a 1. Si esegue l'istruzione AND se lo stato di Carry è uguale a 0.

JR disp – SALTO RELATIVO AL CONTENUTO PRESENTE NEL CONTATORE DI PROGRAMMA



JR disp

18 dd-2

Somma il contenuto del secondo byte del codice oggetto dell'istruzione JR, il contenuto del Contatore di Programma e 2. Carica la somma nel Contatore di Programma. Il salto è misurato dall'indirizzo del codice operativo dell'istruzione, ed ha un campo variabile da –126 a +129 byte. L'Assembler si regola automaticamente per il doppio incremento di PC.

La seguente affermazione in linguaggio assembly è usata per saltare quattro passi avanti l'indirizzo 4000_{16}.

JR $ + 4

Il risultato di questa istruzione ò il seguente:

Locazione	Istruzione
4000	18
4001	02
4002	-
4003	-
4004	- ← nuovo valore di PC

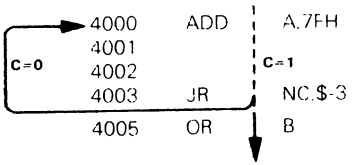
JR NC,disp — SALTO RELATIVO AL CONTENUTO DEL CONTATORE DI PROGRAMMA SE IL FLAG DI CARRY È AZZERATO

JR NC,disp

30 dd-2

Questa istruzione è identica all'istruzione JR disp, tranne che il salto è eseguito solo se lo stato di Carry è uguale a 0; altrimenti si esegue l'istruzione successiva.

Nella seguente sequenza di istruzioni:



Dopo, l'istruzione JR NC,$ — 3, si esegue l'istruzione OR se lo stato di Carry è uguale ad 1. Si esegue l'istruzione ADD se lo stato di Carry è uguale a 0.

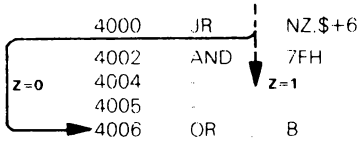
JR NZ, disp — SALTO RELATIVO AL CONTENUTO DEL CONTATORE DI PROGRAMMA SE IL FLAG ZERO È AZZERATO

JR NZ,disp

20 dd-2

Questa istruzione è identica all'istruzione JR disp, tranne che il salto è eseguito solo se lo stato Zero è uguale a 0; altrimenti si esegue l'istruzione successiva.

nella seguente sequenza di istruzioni:



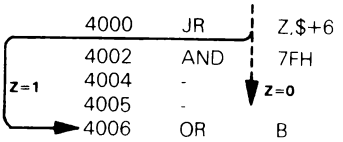
dopo l'istruzione JR NZ,$ + 6, si esegue l'istruzione OR se lo stato di Zero è uguale a 0. Si esegue l'istruzione AND se lo stato Zero è uguale a 1.

JR Z,disp — SALTO RELATIVO AL CONTENUTO DEL CONTATORE DI PROGRAMMA SE IL FLAG ZERO È POSIZIONATO AD 1

JR Z,disp
28 dd-2

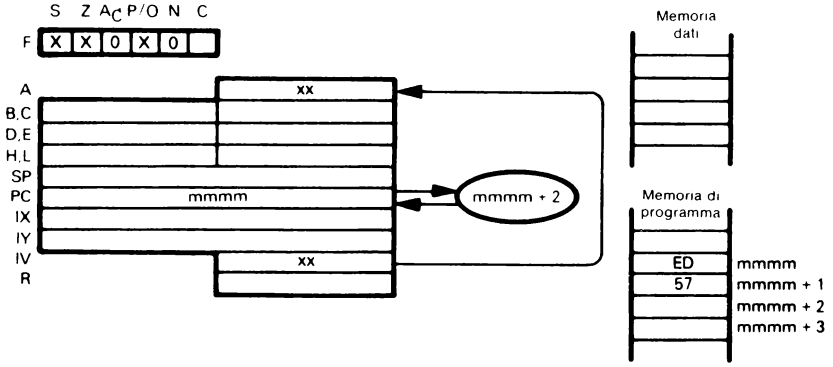
Questa istruzione è identica all'istruzione JR disp, tranne che il salto è eseguito solo se lo stato Zero è uguale ad 1; altrimenti, si esegue l'istruzione successiva.

Nella seguente sequenza di istruzioni:



Dopo l'istruzione JR Z,$ + 6, si esegue l'istruzione OR se lo stato Zero è uguale a 1. Si esegue l'istruzione AND se lo stato Zero è uguale a 0.

LD A,I — LD A,R — SPOSTA IL CONTENUTO DEL VETTORE D'INTERRUZIONE O DEL REGISTRO DI RINFRESCO NELL'ACCUMULATORE



L'illustrazione mostra l'esecuzione di LD A,V:

LD A,V
ED 57

Sposta il contenuto del registro del Vettore d'Interruzione nell'Accumulatore e riflette lo stato di abilitazione dell'interruzione nel flag di Parità/Overflow.

Supponiamo che il registro del Vettore d'Interruzione contenga $7F_{16}$ e che le interruzioni siano disabilitate. Dopo l'esecuzione di

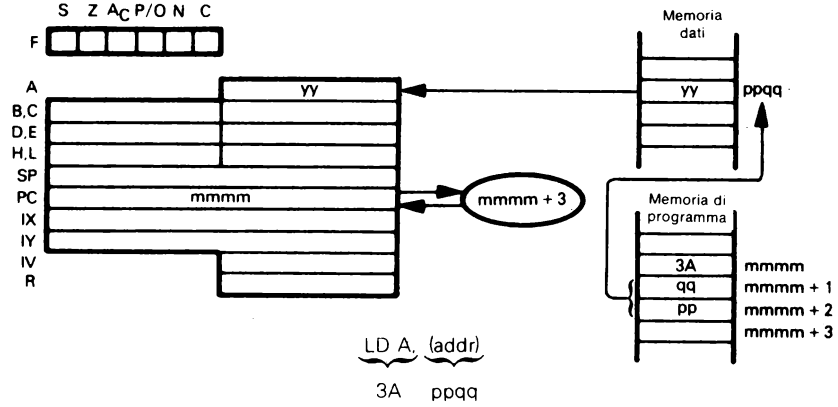
LD A,IV

Il Registro A conterrà $7F_{16}$ e P/O sarà 0.

LD A,R
ED 5F

Sposta il contenuto del registro di Rinfresco nell'Accumulatore. Il valore del flip-flop d'interruzione apparirà nel flag di Parità/Overflow.

LD A,(addr) — CARICA L'ACCUMULATORE DALLA MEMORIA USANDO UN INDIRIZZAMENTO DIRETTO



Carica il contenuto del byte di memoria (indirizzato direttamente dal secondo e dal terzo byte del codice oggetto dell'istruzionie LD A,(addr)) nell'Accumulatore. Supponiamo che il byte di memoria $084A_{16}$ contenga 20_{16}. Dopo l'esecuzione dell'istruzione

```
label   EQU     084AH
—
—
—
LD      A,(label)
```

l'Accumulatore conterrà 20_{16}.

Ricordiamo che EQU è una direttiva Assembler piuttosto che un'istruzione; essa dice all'Assembler di usare il valore a 16 bit $084A_{16}$ dovunque compaia label.

L'istruzione

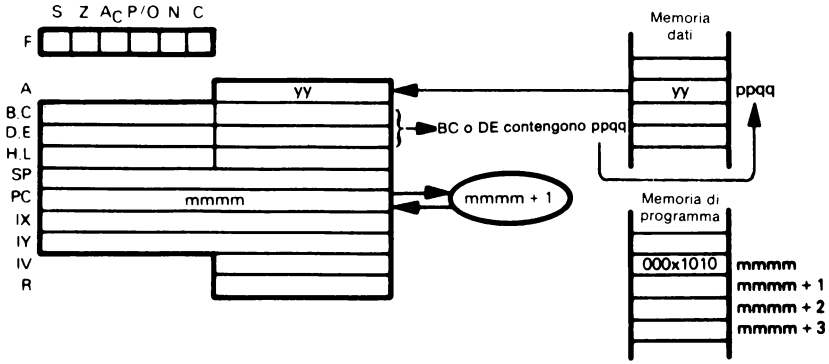
```
LD      A,(label)
```

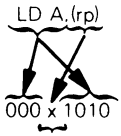
è equivalente alle due istruzioni

```
LD      HL,label
LD      A,(HL)
```

Quando si carica un singolo valore dalla memoria, si preferisce l'istruzione LD A,(label): si usano una sola istruzione a tre byte di programma oggetto per fare ciò che la combinazione LD HL,label, LD A,(HL) fa con due istruzioni e con quattro byte di programma oggetto. Inoltre, la combinazione LD HL,label e LD A ,(HL) usa i registri H ed L che non sono usati da LD A,(label).

LD A,(rp) – CARICA L'ACCUMULATORE DALLA LOCAZIONE DI MEMORIA INDIRIZZATA DALLA COPPIA DI REGISTRI





0 se la coppia di registri è = BC
1 se la coppia di registri è = DE

Carica il contenuto del byte di memoria (indirizzato dalla coppia di registri BC o DE) nell'Accumulatore.

Supponiamo che il registro B contenga 08_{16}, che il registro C contenga $4A_{16}$ e che il byte di memoria $084A_{16}$ contenga $3A_{16}$. Dopo l'esecuzione dell'istruzione

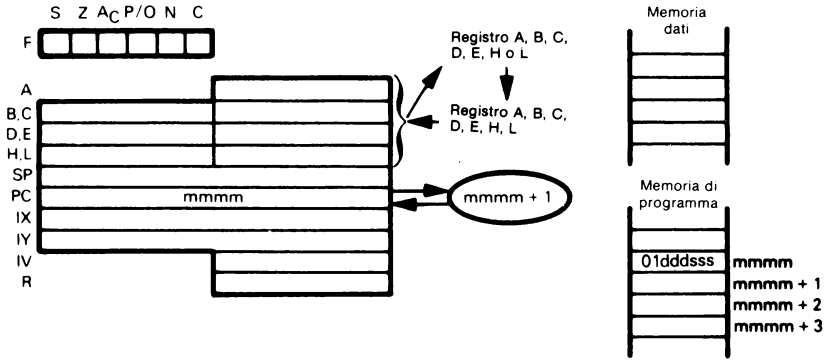
LD A,(BC)

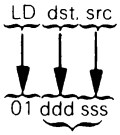
l'Accumulatore conterrà $3A_{16}$.

Normalmente, si useranno insieme LD A,(rp) e LD rp,data, poichè l'istruzione LD rp, data carica un indirizzo di 16 bit nei registri BC o DE come segue:

```
LD      BC,084AH
LD      A,(BC)
```

LD dst,src — SPOSTA IL CONTENUTO DEL REGISTRO SORGENTE NEL REGISTRO DI DESTINAZIONE





000 per dst o src = B
001 per dst o src = C
010 per dst o src = D
011 per dst o src = E
100 per dst o src = H
101 per dst o src = L
111 per dst o src = A

Il contenuto di un qualsiasi registro indicato è caricato in un altro registro.

Per esempio:

LD A,B

Carica il contenuto del Registro B nel Registro A.

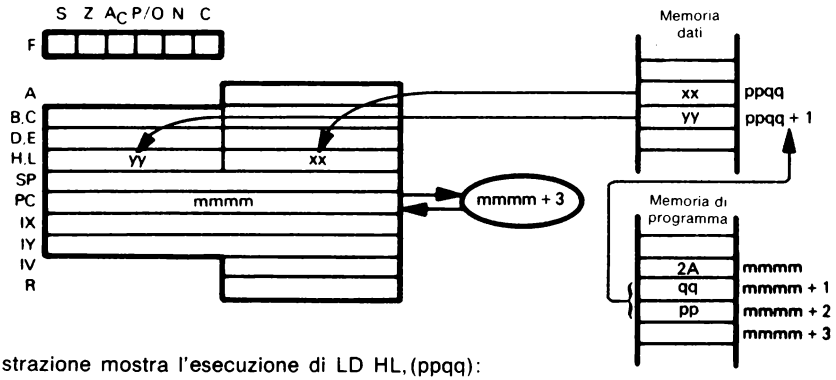
LD L,D

Carica il contenuto del Registro D nel Registro L.

LD C,C

non fa niente, poichè il registro C è stato specificato sia come sorgente che come destinazione.

LD HL,(addr) — CARICA LA COPPIA DI REGISTRI OPPURE
LD rp,(addr) IL REGISTRO INDICE DALLA MEMORIA USANDO
LD IX,(addr) UN INDIRIZZAMENTO DIRETTO
LD IY,(addr)



L'illustrazione mostra l'esecuzione di LD HL,(ppqq):

$$\underbrace{\text{LD HL}}_{\text{2A}}\,\underbrace{\text{,addr}}_{\text{ppqq}}$$

Carica la coppia di registri HL dalla locazione di memoria indirizzata direttamente.

Supponiamo che la locazione di memoria 4004_{16} contenga AD_{16} e che la locazione di memoria 4005_{16} contenga 12_{16}. Dopo l'esecuzione dell'istruzione

LD HL,(4004H)

la coppia di registri HL conterrà $12AD_{16}$.

$$\underbrace{\text{LD rp,}}\,\underbrace{\text{(addr)}}$$

$$\text{ED 01}\,\underbrace{\text{dd}}\,\text{1011 ppqq}$$

00 per rp è la coppia di registri BC
01 per rp è la coppia di registri DE
10 per rp è la coppia di registri HL
11 per rp è lo Stack Pointer

Carica la coppia di registri dalla memoria indirizzata direttamente.

Supponiamo che la locazione di memoria $49FF_{16}$ contenga BE_{16} e che la locazione di memoria $4A00_{16}$ contenga 33_{16}. Dopo l'esecuzione dell'istruzione

LD DE,(49FFH)

la coppia di registri DE conterrà $33BE_{16}$.

$$\underbrace{\text{LD IX,}}\,\underbrace{\text{(addr)}}$$

$$\text{DD 2A ppqq}$$

Carica il registro IX dalla memoria indirizzata direttamente.

Supponiamo che la locazione di memoria $D111_{16}$ contenga FF_{16} e che la locazione di memoria $D112_{16}$ contenga 56_{16}. Dopo l'esecuzione dell'istruzione

LD IX,(D111H)

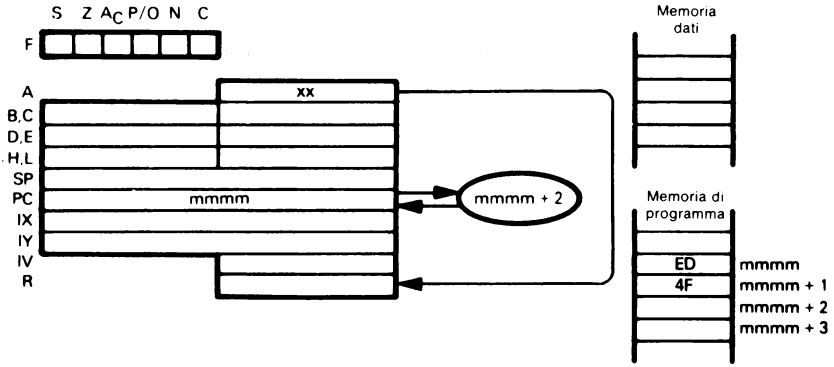
il registro IX conterrà $56FF_{16}$.

$$\underbrace{\text{LD IY,}}_{\text{FD 2A}}\underbrace{\text{(addr)}}_{\text{ppqq}}$$

Carica il registro IY dalla memoria indirizzata direttamente.

Interessa il registro IY invece di IX. Altrimenti è indicata a LD IX,(addr).

LD I,A — CARICA IL VETTORE D'INTERRUZIONE OPPURE
LD R,A IL REGISTRO DI RINFRESCO DELL'ACCUMULATORE



L'illustrazione mostra l'esecuzione di LD R,A:

$$\underbrace{\text{LD R,A}}_{\text{ED 4F}}$$

Carica il registro di Rinfresco dall'Accumulatore.

Supponiamo che l'Accumulatore contenga $7F_{16}$. Dopo l'esecuzione dell'Istruzione

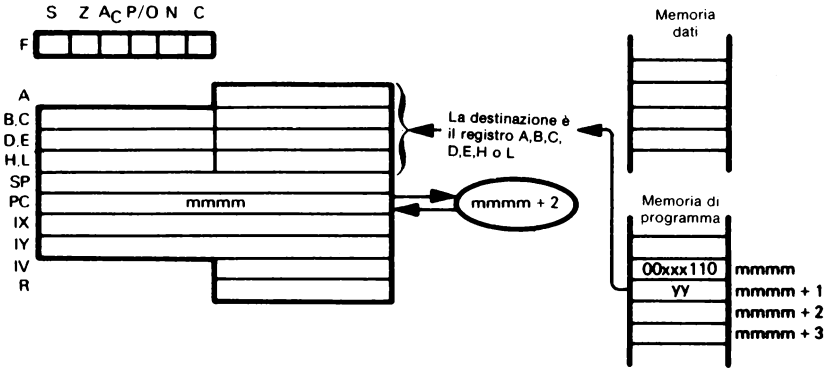
LD R,A

il registro di Rinfresco conterrà $7F_{16}$.

$$\underbrace{\text{LD I,A}}_{\text{ED 47}}$$

Carica il registro del Vettore d'Interruzione dall'Accumulatore.

LD reg,data — CARICA IMMEDIATAMENTE NEL REGISTRO



LD reg,data

00 xxx 110 yy

000 per reg=B
001 per reg=C
010 per reg=D
011 per reg=E
100 per reg=H
101 per reg=L
111 per reg=A

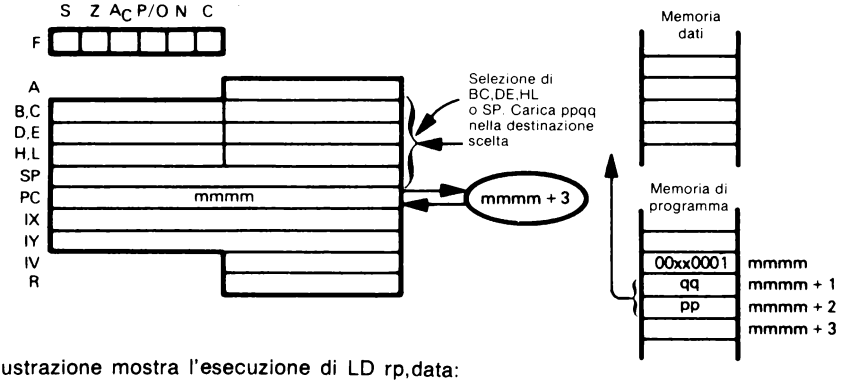
Carica il contenuto del secondo byte del codice oggetto in uno dei registri.

Dopo che si è eseguita l'istruzione

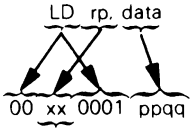
LD A,2AH

si è caricato $2A_{16}$ nell'Accumulatore.

LD rp,data — CARICA UN DATO IMMEDIATO DI 16 BIT
LD IX,data NEL REGISTRO
LD IY,data



L'illustrazione mostra l'esecuzione di LD rp,data:



00 per rp è la coppia di registri BC
01 per rp è la coppia di registri DE
10 per rp è la coppia di registri HL
11 per rp è lo Stack Pointer

Carica il contenuto del secondo e del terzo byte del codice oggetto nella coppia di registri selezionata. Dopo l'esecuzione dell'istruzione

LD SP,217AH

lo Stack Pointer conterrà $217A_{16}$.

LD IX, data
DD 21 ppqq

Carica il contenuto del secondo e del terzo byte del codice oggetto nel registro Indice IX.

LD IY, data
FD 21 ppqq

Carica il contenuto del secondo e del terzo byte del codice oggetto nel registro Indice IY. È da notare che l'istruzione LD rp,data è equivalente a due istruzioni LD reg,data.

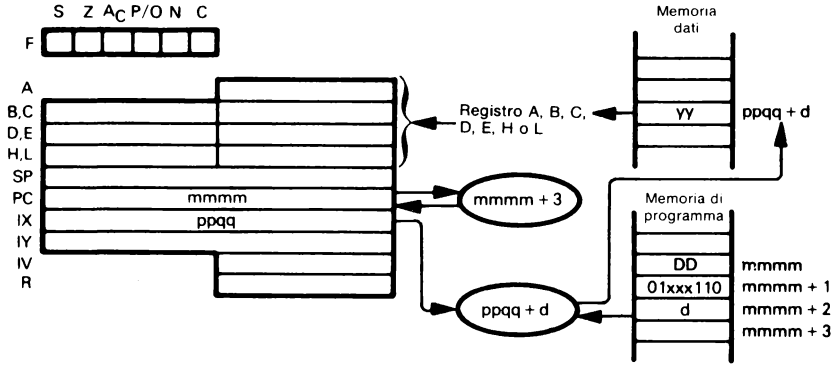
Per esempio:

```
LD      HL,032AH
```

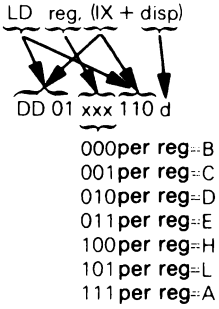
è equivalente a:

```
LD      H,03H
LD      L,2AH
```

LD reg,(HL) – CARICA IL REGISTRO DALLA MEMORIA
LD reg,(IX + disp)
LD reg,(IY + disp)



L'illustrazione mostra l'esecuzione di LD reg, (IX + disp).

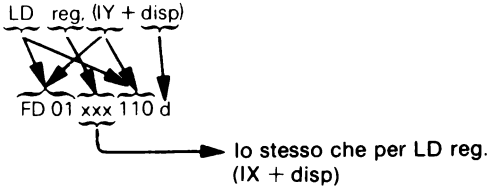


Carica il registro specificato dalla locazione di memoria (specificata dalla somma del contenuto del registro IX e del peso d del dislocamento).

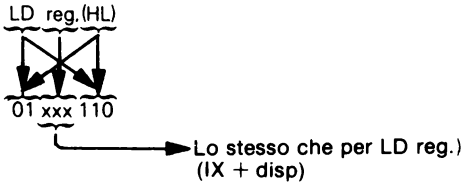
Supponiamo che ppqq = 4004_{16} e che la locazione di memoria 4010_{16} contenga FF_{16}. Dopo l'esecuzione dell'istruzione

LD B(IX + 0CH)

il registro B conterrà FF_{16}.

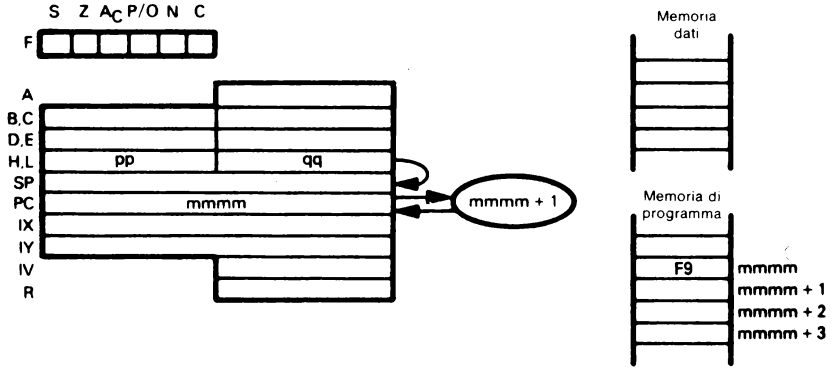


Questa istruzione è identica a LD reg,(IX + disp), tranne che essa usa il registro IY invece del registro IX.



Carica il registro specificato dalla locazione di memoria (specificata dal contenuto della coppia di registri HL).

LD SP,HL – SPOSTA IL CONTENUTO DI HL OPPURE DEL REGISTRO
LD SP,IX INDICE NELLO STACK POINTER
LD SP,IY



L'illustrazione mostra l'esecuzione di LD SP,HL:

$$\underbrace{\text{LD SP,HL}}_{\text{F9}}$$

Carica il contenuto di HL nello Stack Pointer.

Supponiamo che pp = 08_{16} e che qq = $3F_{16}$. Dopo l'esecuzione dell'istruzione

LD SP,HL

lo Stack Pointer conterrà $083F_{16}$.

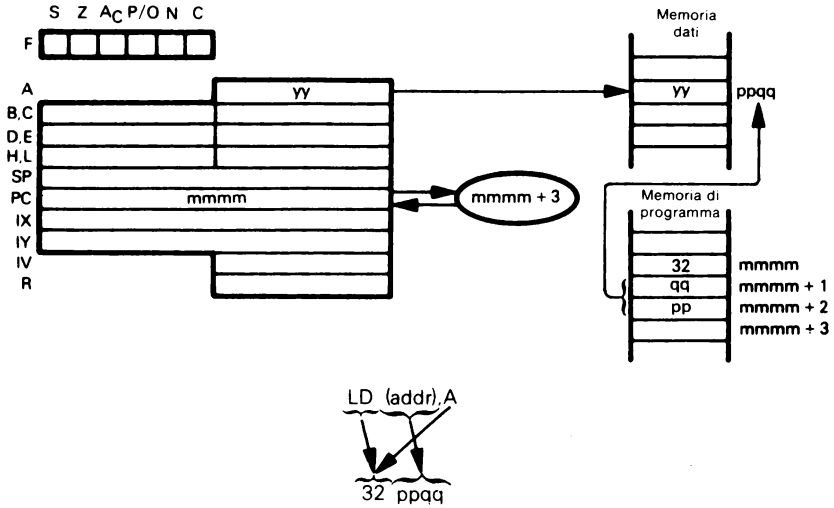
$$\underbrace{\text{LD SP,IX}}_{\text{DD F9}}$$

Carica il contenuto del Registro Indice IX nello Stack Pointer.

$$\underbrace{\text{LD SP,IY}}_{\text{FD F9}}$$

Carica il contenuto del Registro Indice IY nello Stack Pointer.

LD (addr),A — MEMORIZZA L'ACCUMULATORE NELLA MEMORIA USANDO UN INDIRIZZAMENTO DIRETTO



Immagazzina il contenuto dell'Accumulatore nel byte di memoria indirizzato direttamente dal secondo e dal terzo byte del codice oggetto dell'istruzione LD (addr),A.

Supponiamo che l'Accumulatore contenga $3A_{16}$. Dopo l'esecuzione dell'istruzione

```
label    EQU    084AH
         —
         —
         LD     (label),A
```

il byte di memoria $084A_{16}$ conterrà $3A_{16}$.

Ricordiamo che EQU è una direttiva Assembler piuttosto che un'istruzione; essa dice all'Assembler di usare il valore a 16 bit 084AH ogni volta che appare la parola «label».

L'istruzione

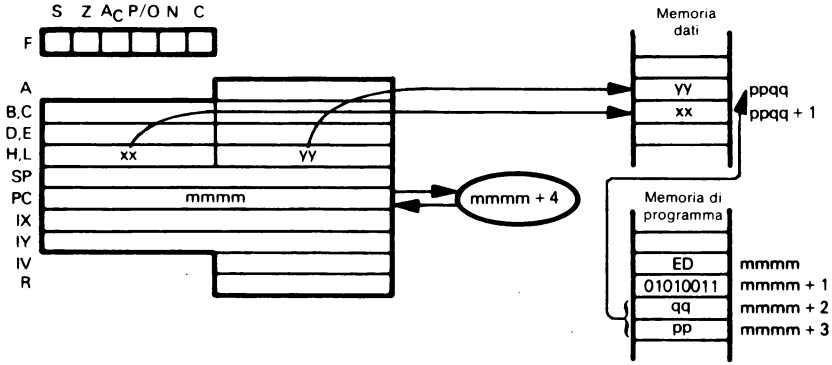
```
LD (addr),A
```

è equivalente alle due istruzioni

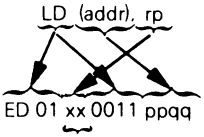
```
LD H,label
LD (HL),A
```

Quando si immagazzina in memoria un singolo valore dati, si preferisce l'istruzione LD (label),A, perchè essa usa una sola istruzione e tre byte di programma oggetto per fare ciò che la combinazione LD H,label ed LD (HL),A fa in due istruzioni e in quattro byte di programma oggetto. Inoltre, la combinazione LD H,label ed LD (HL),A usa i registri H ed L, mentre l'istruzione LD (label),A no.

LD (addr),HL – IMMAGAZZINA LA COPPIA DI REGISTRI
LD (addr),rp O IL REGISTRO INDICE IN MEMORIA USANDO
LD (addr),xy UN INDIRIZZAMENTO DIRETTO



L'illustrazione mostra l'esecuzione di LD (ppqq),DE:



00 per rp è la coppia di registri BC
01 per rp è la coppia di registri DE
10 per rp è la coppia di registri HL
11 per rp è lo Stack Pointer

Immagazzina in memoria il contenuto della coppia di registri specificata. Il terzo e il quarto byte del codice oggetto danno l'indirizzo della locazione di memoria dove si deve scrivere il byte di ordine minore. Il byte di ordine maggiore è scritto nella locazione di memoria successiva della sequenza.

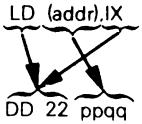
Supponiamo che la coppia di registri BC contenga $3C2A_{16}$. Dopo l'esecuzione della istruzione

```
label    EQU      084AH
—
—
—
LD       (label),BC
```

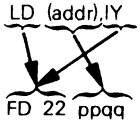
il byte di memoria $084A_{16}$ conterrà $2A_{16}$. Il byte di memoria $084B_{16}$ conterrà $3C_{16}$.

Ricordiamo che EQU è una direttiva Assembler piuttosto che un'istruzione; essa dice all'Assembler di usare il valore a 16 bit $084A_{16}$ ogni volta che appare la parola «label».

Questa è una versione a tre byte di LD (addr), che specifica direttamente HL come coppia di registri sorgente.

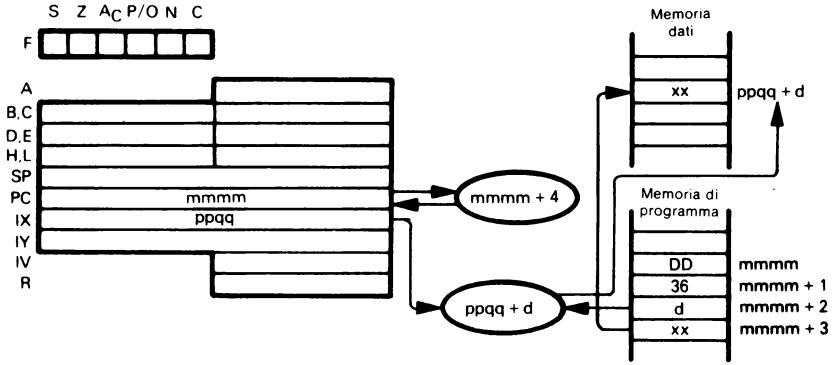


Immagazzina il contenuto del registro Indice IX in memoria. Il terzo e il quarto byte del codice oggetto danno l'indirizzo della locazione di memoria dove si deve scrivere il byte di ordine minore. Il byte di ordine maggiore è scritto nella locazione di memoria successiva della sequenza.



Questa istruzione è identica all'istruzione LD (addr),IX, tranne che essa usa il registro IY invece del registro IX.

LD (HL),data – CARICA IMMEDIATAMENTE NELLA MEMORIA
LD (IX + disp), data
LD (IY + disp), data



L'illustrazione mostra l'esecuzione di LD (IX + d),xx:

$$\underbrace{\text{LD (IX}}_{\text{DD 36}}\underbrace{\text{+disp)}}_{\text{d}}\text{,}\underbrace{\text{data}}_{\text{xx}}$$

Carica immediatamente nella locazione di memoria indicata dall'indirizzamento relativo alla base.

Supponiamo che ppqq = 5400_{16}. Dopo l'esecuzione dell'istruzione

LD (IX + 9),FAH

la locazione di memoria 5409_{16} conterrà FA_{16}.

$$\underbrace{\text{LD (IY}}_{\text{FD 36}}\underbrace{\text{+disp)}}_{\text{d}}\text{,}\underbrace{\text{data}}_{\text{xx}}$$

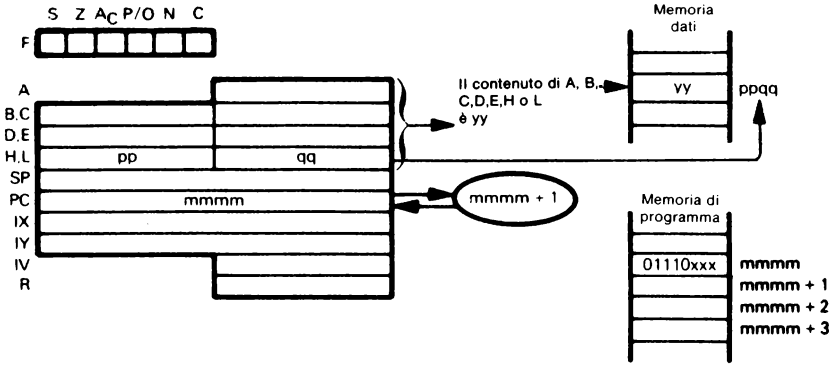
Questa istruzione è identica a LD (IX + disp),data, ma essa usa il registro IY invece del registro IX.

$$\underbrace{\text{LD (HL)}}_{\text{36}}\text{,}\underbrace{\text{data}}_{\text{xx}}$$

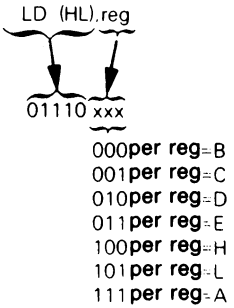
Carica immediatamente nella locazione di memoria (specificata dal contenuto della coppia di registri HL).

Le istruzioni di Caricamento Immediato in Memoria sono molto meno usate delle istruzioni di Caricamento Immediato in Registri.

LD (HL),reg – CARICA LA MEMORIA DA UN REGISTRO
LD (IX + disp),reg
LD (IY + disp),reg



L'illustrazione mostra l'esecuzione di LD (HL),reg:

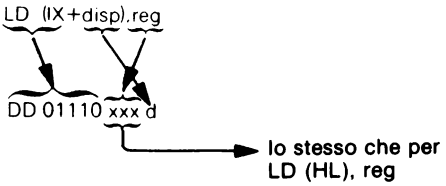


Carica la locazione di memoria (specificata dal contenuto della coppia di registri HL) dal registro specificato.

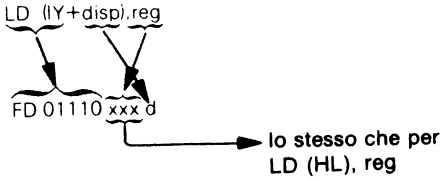
Supponiamo che ppqq = 4500_{16} e che il Registro C contenga $F9_{16}$. Dopo l'esecuzione dell'istruzione

LD (HL),C

la locazione di memoria 4500_{16} conterrà $F9_{16}$.

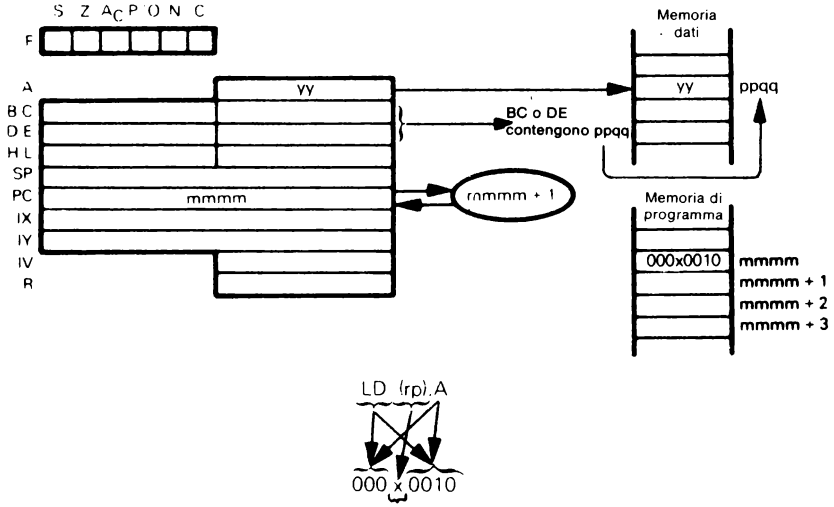


Carica la locazione di memoria (specificata dalla somma del contenuto del registro IX e del valore d del dislocamento) dal registro specificato.



Questa istruzione è identica a LD (IX + disp),reg, tranne che essa usa il registro IY invece del registro IX.

LD (rp),A – CARICA L'ACCUMULATORE NELLA LOCAZIONE DI MEMORIA INDIRIZZATA DALLA COPPIA DI REGISTRI



Immagazzina l'Accumulatore nel byte di memoria indirizzato dalla coppia di registri BC o DE.

Supponiamo che la coppia di registri BC contenga $084A_{16}$ e che l'Accumulatore contenga $3A_{16}$. Dopo l'esecuzione dell'istruzione

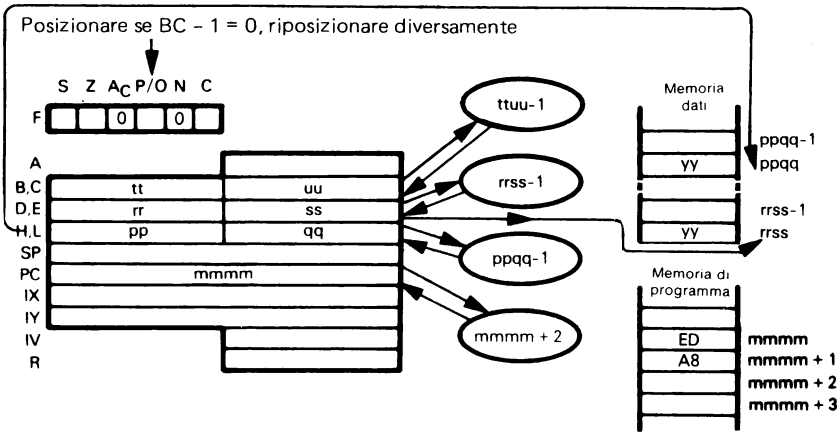
LD (BC),A

il byte di memoria $084A_{16}$ conterrà $3A_{16}$.

Le istruzioni LD (rp),A e LD rp,data saranno normalmente usate insieme, poichè l'istruzione LD rp,data carica un indirizzo a 16 bit nei registri BC o DE come segue:

```
LD BC,084AH
LD (BC),A
```

LDD — TRASFERISCE DATI TRA LOCAZIONI DI MEMORIA. DECREMENTA GLI INDIRIZZI DELLA DESTINAZIONE E DELLA SORGENTE



LDD
ED A8

Trasferisce un byte del dato dalla locazione di memoria indirizzata dalla coppia di registri HL alla locazione di memoria indirizzata dalla coppia di registri DE. Decrementa il contenuto della coppia di registri BC, DE e HL.

Supponiamo che la coppia di registri BC contenga $004F_{16}$, che DE contenga 4545_{16}, che HL contenga 2012_{16} e che la locazione di memoria 2012_{16} contenga 18_{16}. Dopo l'esecuzione dell'istruzione

LDD

la locazione di memoria 4545_{16} conterrà 18_{16}, la coppia di registri BC conterrà $004E_{16}$, DE conterrà 4544_{16} e HL conterrà 2011_{16}.

LDDR — TRASFERISCE DATI TRA LOCAZIONI DI MEMORIA FINCHÈ IL CONTATORE DEI BYTE NON SIA ZERO. DECREMENTA GLI INDIRIZZI DELLA DESTINAZIONE E DELLA SORGENTE

LDDR
ED B8

Questa istruzione è identica a LDD, tranne che essa si ripete finchè la coppia di registri BC non contenga zero. Dopo ogni trasferimento di un dato, si riconosceranno le interruzioni e si eseguiranno due cicli di rinfresco.

Supponiamo di avere i seguenti contenuti in memoria e nelle coppie di registri:

Registro/Contenuto		Locazione/Contenuto	
HL	2012_{16}	2012_{16}	18_{16}
DE	4545_{16}	2011_{16}	AA_{16}
BC	0003_{16}	2010_{16}	25_{16}

Dopo l'esecuzione di

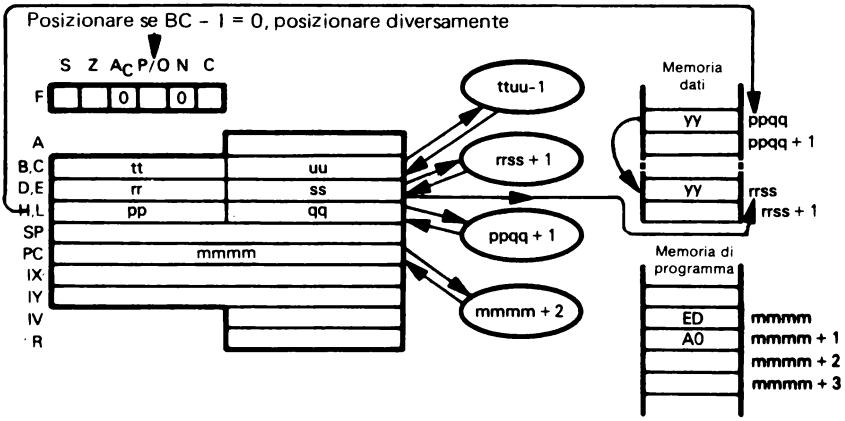
LDDR

le coppie di registri e le locazioni di memoria avranno i contenuti seguenti:

Registro/Contenuto		Locazione/Contenuto		Locazione/Contenuto	
HL	2009_{16}	2012_{16}	18_{16}	4545_{16}	18_{16}
DE	4542_{16}	2011_{16}	AA_{16}	4544_{16}	AA_{16}
BC	0000_{16}	2010_{16}	25_{16}	4543_{16}	25_{16}

Questa istruzione è estremamente utile per trasferire blocchi di dati da un'area di memoria ad un'altra.

LDI — TRASFERISCE DATI TRA LOCAZIONI DI MEMORIA. INCREMENTA GLI INDIRIZZI DELLA DESTINAZIONE E DELLA SORGENTE



LDI

ED A0

Trasferisce un byte dalla locazione di memoria indirizzata dalla coppia di registri HL nella locazione di memoria indirizzata dalla coppia di registri DE. Incrementa il contenuto delle coppie di registri HL e DE. Decrementa il contenuto della coppia di registri BC.

Supponiamo che la coppia di registri BC contenga $004F_{16}$, che DE contenga 4545_{16}, che HL contenga 2012_{16} e che la locazione di memoria 2012_{16} contenga 18_{16}. Dopo l'esecuzione dell'istruzione

LDI

la locazione di memoria 4545_{16} conterrà 18_{16}, la coppia di registri BC conterrà $004E_{16}$, DE conterrà 4546_{16} e HL conterrà 2013_{16}.

LDIR – TRASFERISCE DATI TRA LOCAZIONI DI MEMORIA FINCHÈ IL CONTATORE DEI BYTE NON SIA ZERO. INCREMENTA GLI INDIRIZZI DELLA DESTINAZIONE E DELLA SORGENTE

LDIR
ED B0

Questa istruzione è identica a LDI, tranne che essa si ripete finchè la coppia di registri BC non contiene zero. Dopo ogni trasferimento di un dato, si riconosceranno le interruzioni e si eseguiranno due cicli di rinfresco.

Supponiamo di avere i seguenti contenuti nella memoria e nelle coppie di registri:

Registro/Contenuto		Locazione/Contenuto	
HL	2012_{16}	2012_{16}	18_{16}
DE	4545_{16}	2013_{16}	CD_{16}
BC	0003_{16}	2014_{16}	$F0_{16}$

Dopo l'esecuzione di

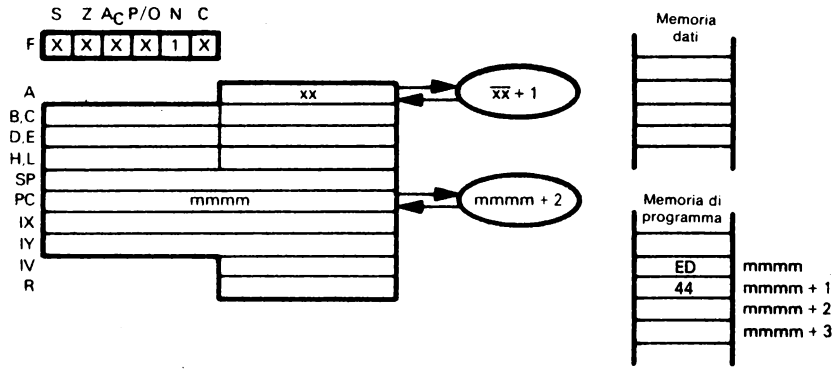
LDIR

le coppie di registri e la memoria avranno i seguenti contenuti:

Registro/Contenuto		Locazione/Contenuto		Locazione/Contenuto	
HL	2015_{16}	2012_{16}	18_{16}	4545_{16}	18_{16}
DE	4548_{16}	2013_{16}	CD_{16}	4546_{16}	CD_{16}
BC	0000_{16}	2014_{16}	$F0_{16}$	4547_{16}	$F0_{16}$

Questa istruzione è estremamente utile per trasferire blocchi di dati da un'area di memoria ad un'altra.

NEG — FA IL COMPLEMENTO A DUE DEL CONTENUTO DELL'ACCUMULATORE



Fa il complemento a due del contenuto dell'Accumulatore. È lo stesso che sottrae il contenuto dell'Accumulatore da zero. Il risultato è il complemento a due. 80H rimarrà immutato.

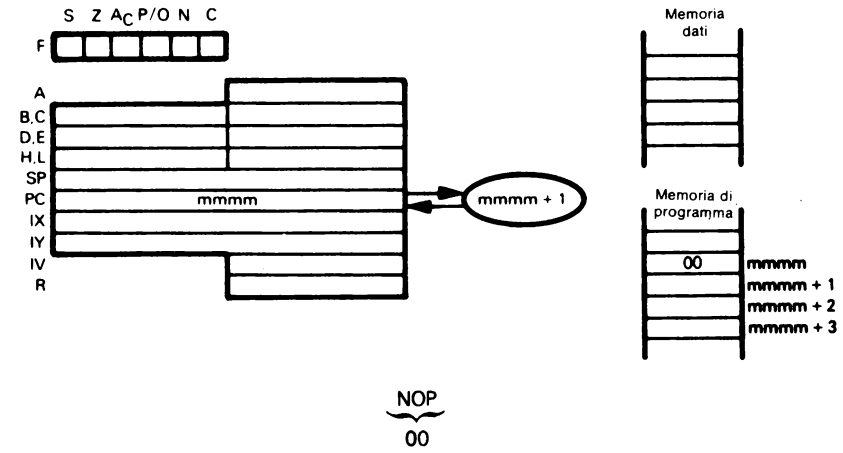
Supponiamo che xx = $5A_{16}$. Dopo l'esecuzione dell'istruzione

NEG

l'Accumulatore conterrà $A6_{16}$.

5A	=	0 1 0 1 1 0 1 0
Complemento a due	=	1 0 1 0 0 1 1 0

NOP – NESSUNA OPERAZIONE



Questa è un'istruzione ad un solo byte che non effettua nessuna operazione, tranne che incrementare il Contatore di Programma e continuare il rinfresco della memoria. Questa istruzione è presente per parecchi motivi:

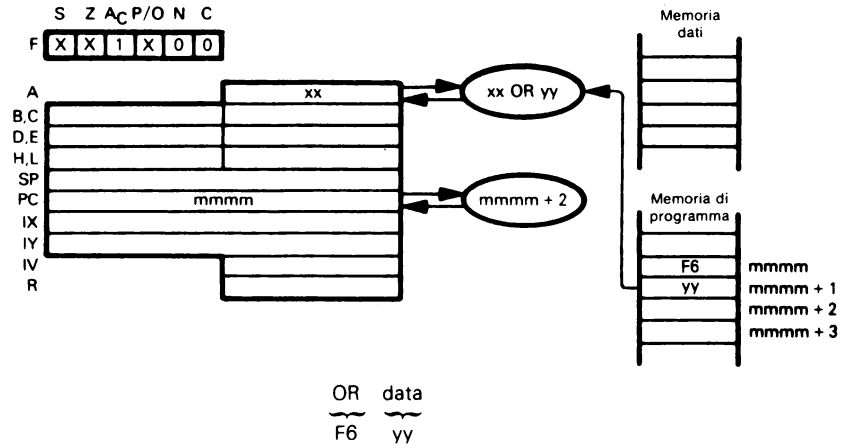
1) Un errore di programma che riporta un codice oggetto da una memoria non esistente riporta 00. È una buona idea per assicurarsi che l'errore di programma più comune non farà niente.

2) L'istruzione NOP permette di dare un'etichetta ad un byte di codice oggetto:
   HERE NOP

3) Per tempi di ritardo a regolazione fine. Ogni istruzione NOP somma il ritardo quattro cicli di clock.

   NOP non è un'istruzione molto utile o usata frequentemente.

OR data — OR IMMEDIATO CON L'ACCUMULATORE



Fa l'OR dell'Accumulatore col contenuto del secondo byte del codice oggetto della istruzione

Supponiamo che xx = $3A_{16}$. Dopo l'esecuzione dell'istruzione

OR 7CH

l'Accumulatore conterrà $7E_{16}$.

```
3A = 0011 1010
7C = 0111 1100
     0111 1110
```

0 posiziona S a 0

Sei bit a 1, posizionano P/O ad 1

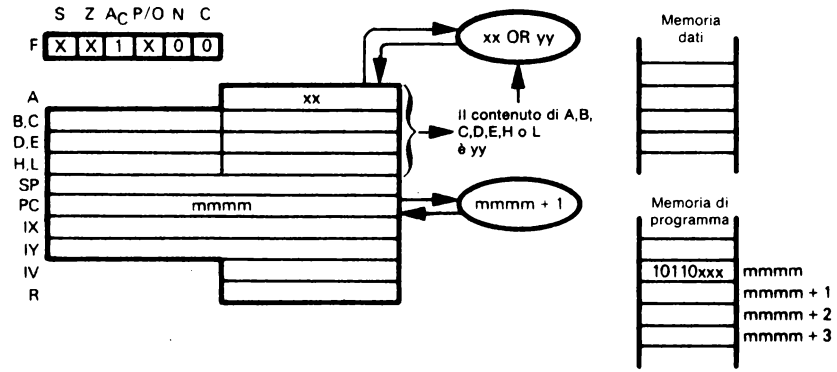
Risultato non zero, posiziona Z a 0

Questa è un'istruzione logica di programma; essa è spesso usata per porre dei bit «on». Per esempio l'istruzione

OR 80H

Posizionerà incondizionatamente ad 1 bit di ordine maggiore dell'Accumulatore.

OR reg — OR DEL REGISTRO CON L'ACCUMULATORE



OR reg

10110 xxx

000 per reg=B
001 per reg=C
010 per reg=D
011 per reg=E
100 per reg=H
101 per reg=L
111 per reg=A

Fa l'OR logico del contenuto dell'Accumulatore col contenuto del Registro A, B, C, D, E, H o L. Immagazzina il risultato nell'Accumulatore.

Supponiamo che xx = $E3_{16}$ e che il Registro E contenga $A8_{16}$. Dopo l'esecuzione della istruzione

OR E

l'Accumulatore conterrà EB_{16}.

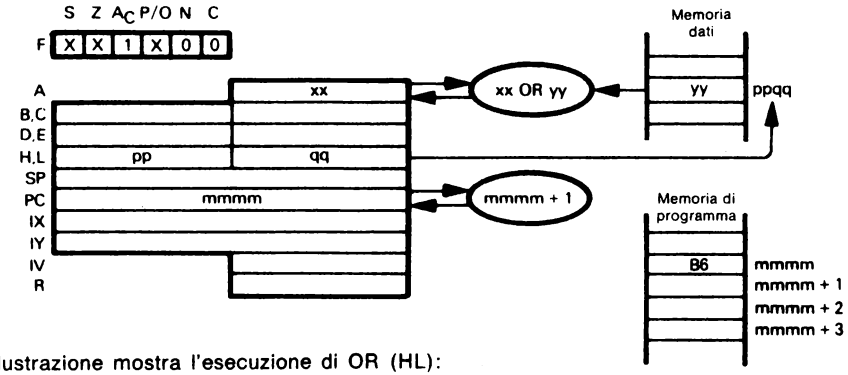
```
E3 = 1110 0011
A8 = 1010 1000
     ---------
     1110 1011
```

1 posiziona S a 1

Sei bit ad 1 posizionano P/O ad 1

Risultato non zero, posiziona Z a 0

OR (HL) – OR DELLA MEMORIA CON L'ACCUMULATORE
OR (IX + disp)
OR (IY + disp)



L'illustrazione mostra l'esecuzione di OR (HL):

$$\underbrace{\text{OR (HL)}}_{\text{B6}}$$

Fa l'OR del contenuto della locazione di memoria (specificata dal contenuto della coppia di registri HL) con l'Accumulatore.

Supponiamo che xx = $E3_{16}$, ppqq = 4000_{16} e che la locazione di memoria 4000_{16} contenga $A8_{16}$. Dopo l'esecuzione dell'istruzione

OR (HL)

l'Accumulatore conterrà EB_{16}.

```
E3 = 1110  0011
A8 = 1010  1000
     ----------
     1110  1011
```

1 posiziona S a 1

Sei bit ad 1 posizionano P/O ad 1

Risultato non zero, posiziona Z a 0

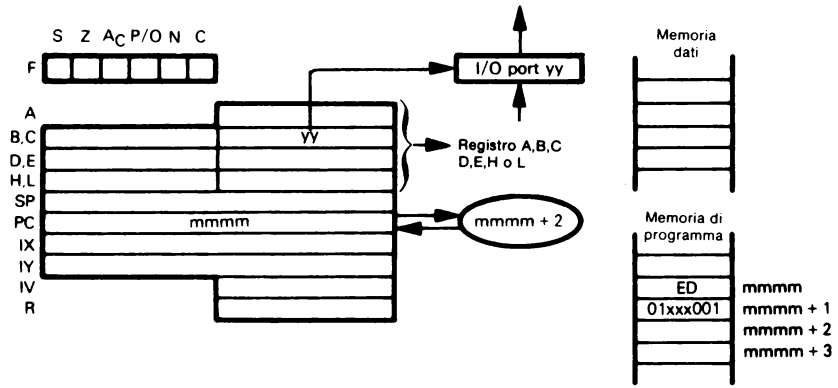
$$\underbrace{\text{OR (IX}}_{\text{DD B6}}\underbrace{\text{+disp)}}_{\text{d}}$$

Fa l'OR del contenuto della locazione di memoria (specificata dalla somma del contenuto del registro IX e del valore d del dislocamento) con l'Accumulatore.

$$\underbrace{\text{OR (IY}}_{\text{FD B6}}\underbrace{\text{+disp)}}_{\text{d}}$$

Questa istruzione è identica a OR (IX + disp), tranne che essa usa il registro IY invece del registro IX.

OUT (C),reg — USCITA DA UN REGISTRO



OUT (C),reg

ED 01 xxx 001

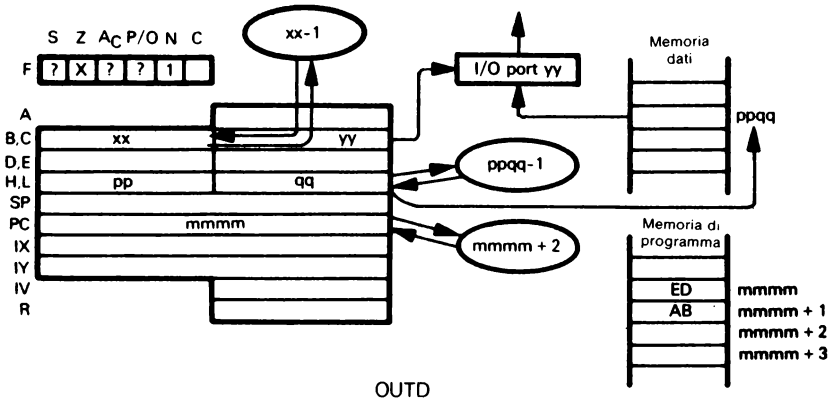
000 per reg=B
001 per reg=C
010 per reg=D
011 per reg=E
100 per reg=H
101 per reg=L
111 per reg=A

Supponiamo che yy = $1F_{16}$ e che il contenuto di H sia AA_{16}. Dopo l'esecuzione di

OUT (C),H

AA_{16} sarà nel buffer della porta di I/O $1F_{16}$.

OUTD — USCITA DALLA MEMORIA, DECREMENTA L'INDIRIZZO



OUTD
ED AB

Uscita dalla locazione di memoria specificata da HL verso la porta di I/O indirizzata dal Registro C. Si decrementano i registri B e HL.

Supponiamo che xx = $0A16_{16}$, yy = FF_{16}, ppqq = 5000_{16} e che la locazione di memoria 5000_{16} contenga 77_{16}. Dopo l'esecuzione dell'istruzione

OUTD

nel buffer della porta di I/O FF_{16} sarà contenuto 77_{16}. Il registro B conterrà 09_{16} e la coppia di registri HL conterrà $4FFF_{16}$.

OTDR — USCITA DALLA MEMORIA. DECREMENTA L'INDIRIZZO, CONTINUA FINCHÈ IL REGISTRO B = 0

OTDR
ED BB

OTDR è identica a OUTD, ma si ripete finchè il Registro B non contenga 0.

Supponiamo che il Registro B contenga 03_{16}, che il Registro C contenga FF_{16} e che HL contenga 5000_{16}. Le locazioni di memoria tra $4FFE_{16}$ e 5000_{16} contengano:

Locazione	Contenuto
$4FFE_{16}$	CA_{16}
$4FFF_{16}$	$1B_{16}$
5000_{16}	$F1_{16}$

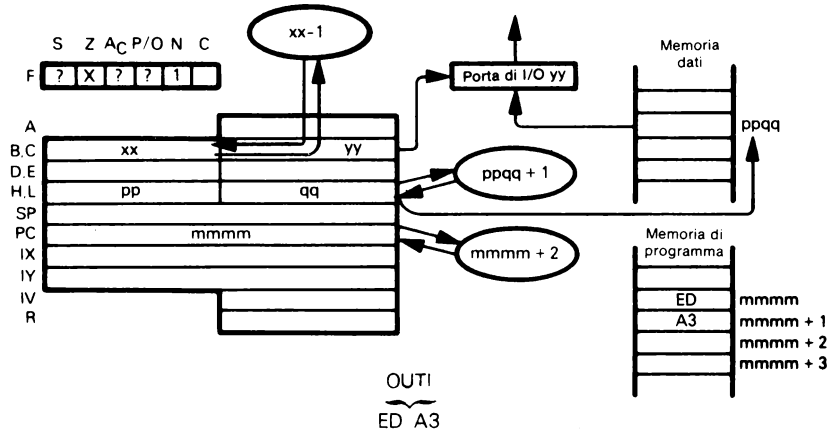
Dopo l'esecuzione di

OTDR

la coppia di registri HL conterrà $4FFD_{16}$, il Registro B conterrà zero e nella porta di I/O FF_{16} ci sarà stata scritta la sequenza $F1_{16}$, $1B_{16}$, CA_{16}.

Questa istruzione è molto utile per trasferire blocchi di dati dalla memoria a dispositivi di uscita.

OUTI — USCITA DALLA MEMORIA. INCREMENTO DELL'INDIRIZZO



Uscita dalla locazione di memoria specificata da HL verso la porta di I/O indirizzata dal Registro C. Si decrementa il Registro B e si incrementa la coppia di registri HL.

Supponiamo che xx = $0A_{16}$, yy = FF_{16}, ppqq = 5000_{16} e che la locazione di memoria 5000_{16} contenga 77_{16}. Dopo l'esecuzione dell'istruzione

OUTI

il buffer della porta di I/O FF_{16} conterrà 77_{16}. Il registro B conterrà 09_{16} e la coppia di registri HL conterrà 5001_{16}.

OTIR — USCITA DALLA MEMORIA. INCREMENTA L'INDIRIZZO, CONTINUA FINCHÈ IL REGISTRO B = 0

OTIR
ED B3

OTIR è identica a OUTI, tranne che essa si ripete finchè il Registro B non contiene 0.

Supponiamo che il Registro B contenga 04_{16}, che il Registro C contenga FF_{16} e che HL contenga 5000_{16}. Le locazioni di memoria da 5000_{16} a 5003_{16} contengono:

Locazione	Contenuto
5000_{16}	CA_{16}
5001_{16}	$1B_{16}$
5002_{16}	$B1_{16}$
5003_{16}	AD_{16}

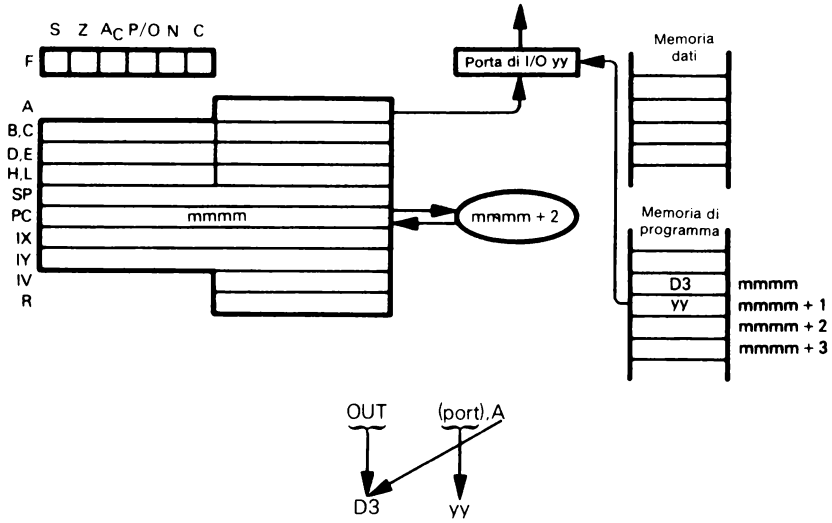
Dopo l'esecuzione di

OTIR

la coppia di registri HL conterrà 5004_{16}, il Registro B conterrà zero e nella porta di I/O FF_{16} sarà scritta la sequenza CA_{16}, $1B_{16}$, $B1_{16}$ e AD_{16}.

Questa istruzione è molto utile per trasferire blocchi di dati dalla memoria ad un dispositivo di uscita.

OUT (port),A — USCITA DALL'ACCUMULATORE



Fa uscire il contenuto dell'Accumulatore verso la porta di I/O identificata dal secondo byte del codice oggetto dell'istruzione OUT.

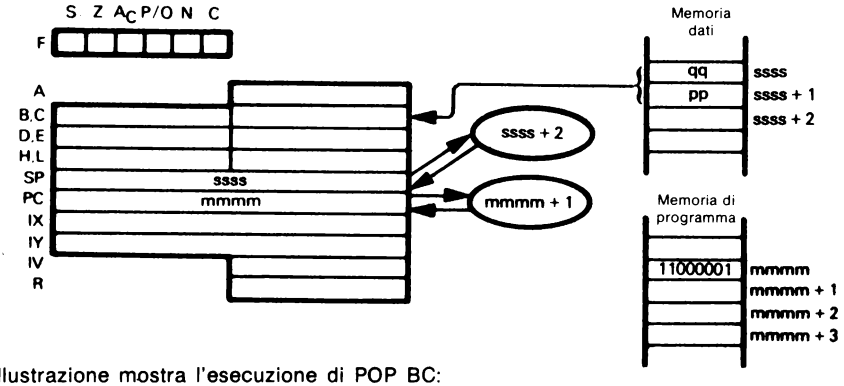
Supponiamo che l'Accumulatore contenga 36_{16}. Dopo l'esecuzione dell'istruzione

OUT (1AH),A

il buffer della porta di I/O $1A_{16}$ conterrà 36_{16}.

L'istruzione OUT non influenza nessuno stato. L'uso dell'istruzione OUT è molto dipendente dall'hardware. Gli indirizzi validi per la porta di I/O sono determinati dal modo con cui si è implementata la logica di I/O. È inoltre possibile progettare un sistema a microcalcolatore che acceda alla logica esterna usando istruzioni di riferimento alla memoria con specifici indirizzi di memoria. Le istruzioni OUT sono usate frequentemente, in modo speciale per controllare la logica del microcalcolatore esterno alla CPU.

POP rp — LETTURA DALLA SOMMITÀ DELLO STACK
POP IX
POP IY



L'illustrazione mostra l'esecuzione di POP BC:

POP rp

11 xx 0001

00 per rp è la coppia di registri BC
01 per rp è la coppia di registri DE
10 per rp è la coppia di registri HL
11 per rp è la coppia di registri A e F

Estrae (POP) i due byte in cima allo stack e li mette nella coppia di registri indicata.

Supponiamo che qq = 01_{16} e che pp = $2A_{16}$. L'esecuzione di

POP HL

carica 01_{16} nel registro L e $2A_{16}$ nel registro H. L'esecuzione dell'istruzione

POP AF

carica 01 nei flag degli stati e $2A_{16}$ nell'Accumulatore. In tale modo, lo stato di Carry sarà posizionato ad 1 e gli altri stati saranno azzerati.

POP IX

DD E1

Estrae (POP) i due byte in cima allo stack e li mette nel registro IX.

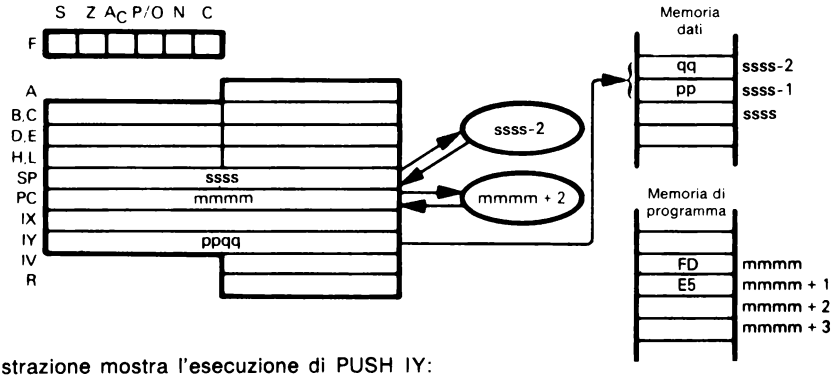
POP IY

FD E1

Estrae (POP) i due byte in cima allo stack e li mette nel registro IY.

L'istruzione POP è grandemente usata per ripristinare il contenuto dei registri e degli stati che sono stati salvati nello stack; per esempio, durante il servizio di una interruzione.

PUSH rp – SCRITTURA SULLA SOMMITÀ DELLO STACK
PUSH IX
PUSH IY



L'illustrazione mostra l'esecuzione di PUSH IY:

$$\underbrace{\text{PUSH IY}}_{\text{FD E5}}$$

Spinge (PUSH) il contenuto del registro IY sulla sommità dello stack.

Supponiamo che il registro IY contenga $45FF_{16}$; l'esecuzione dell'istruzione

PUSH IY

carica 45_{16}; poi FF_{16} sulla sommità dello stack.

$$\underbrace{\text{PUSH IX}}_{\text{DD E5}}$$

Spinge (PUSH) il contenuto del registro IX sulla sommità dello stack.

$$\underbrace{\text{PUSH rp}}$$

$$\underbrace{11}\ \underbrace{xx}\ \underbrace{0101}$$

00 per rp è la coppia di registri BC
01 per rp è la coppia di registri DE
10 per rp è la coppia di registri HL
11 per rp è la coppia di registri A e F

Spinge (PUSH) il contenuto della coppia di registri indicata sulla sommità dello stack.

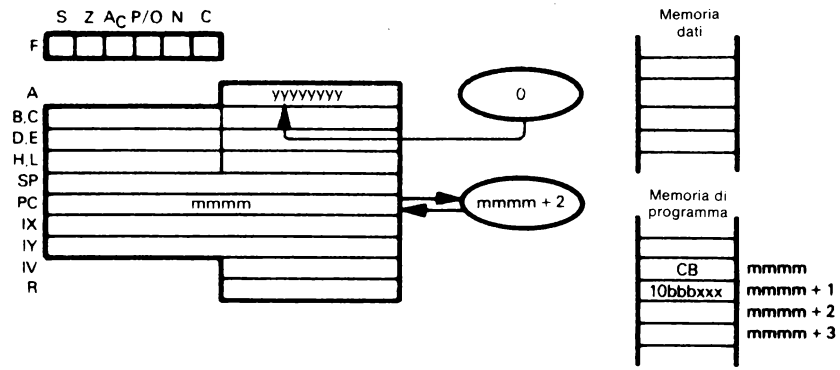
L'esecuzione dell'istruzione

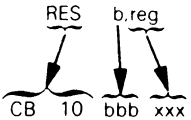
PUSH AF

carica l'Accumulatore e poi i flag degli stati sulla sommità dello stack.

L'istruzione PUSH è grandemente usata per salvare il contenuto dei registri e degli stati; per esempio, prima di servire un'interruzione.

RES b,reg — AZZERA IL BIT DEL REGISTRO INDICATO





Bit	bbb	xxx	Registro
0	000	000	B
1	001	001	C
2	010	010	D
3	011	011	E
4	100	100	H
5	101	101	L
6	110	111	A
7	111		

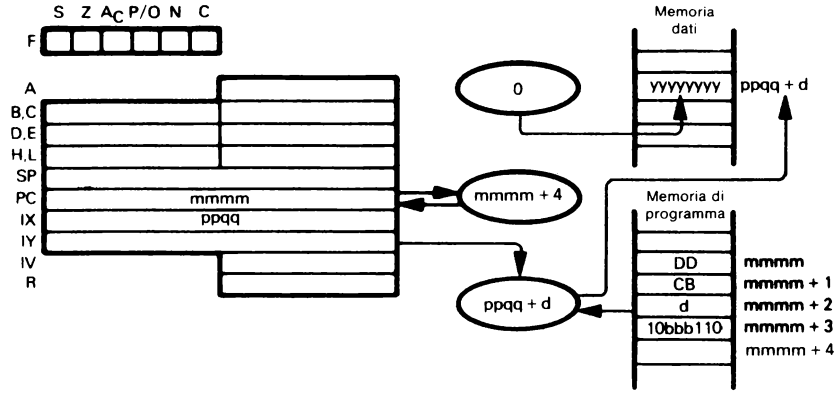
Azzera il bit indicato nel registro specificato.

Dopo l'esecuzione dell'istruzione

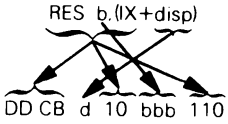
RES 6,H

sarà azzerato il bit 6 nel Registro H. (Il bit 0 è il bit meno significativo).

RES b,(HL) – AZZERA IL BIT b DELLA POSIZIONE
RES b,(IX + disp) DI MEMORIA INDICATA
RES b,(IY + disp)



L'illustrazione mostra l'esecuzione di RES b,(IX + disp). Il bit 0 è il bit meno significativo.



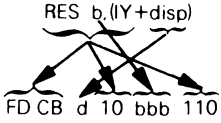
bbb	Bit Azzerato
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Azzera il bit indicato nella locazione di memoria indicata dalla somma del Registro Indice IX e d.

Supponiamo che IX contenga 4110_{16}. Dopo l'esecuzione dell'istruzione

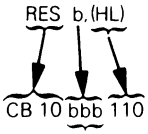
RES 0,(IX + 7)

il bit 0 della locazione di memoria 4117_{16} sarà 0.



bbb è lo stesso che per RES b., (IX + disp)

Questa istruzione è identica a RES b,(IX + disp), tranne che essa usa il registro IY invece del registro IX.



bbb è lo stesso che per RES b, (IX + disp)

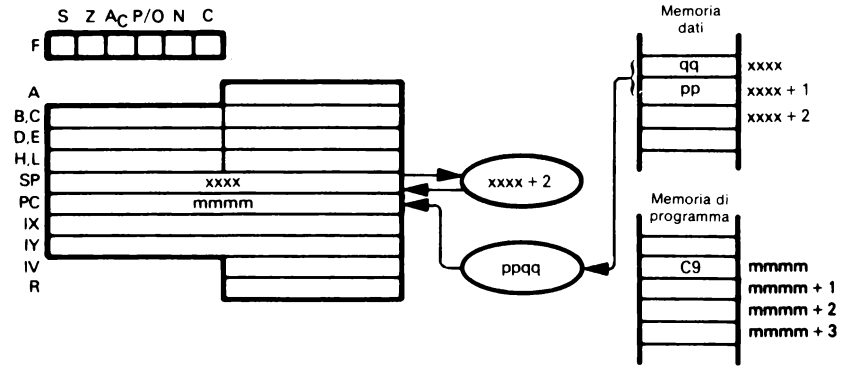
Azzera il bit indicato nella locazione di memoria indicata da HL.

Supponiamo che HL contenga 4444_{16}. Dopo l'esecuzione di

RES 7,(HL)

il bit 7 della locazione di memoria 4444_{16} sarà 0.

RET — RITORNO DA SOTTOPROGRAMMA

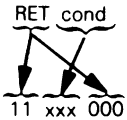


RET
C9

Sposta il contenuto dei due byte in cima allo stack nel Contatore di Programma; questi due byte forniscono l'indirizzo dell'istruzione che si deve eseguire successivamente. Il contenuto precedente del Contatore di programma è perduto. Incrementa lo Stack Pointer di 2, per indirizzare nuovamente la cima dello stack.

Ogni sottoprogramma deve contenere almeno una istruzione di Return (o un Return condizionato); questa è l'ultima istruzione eseguita nel sottoprogramma e provoca il ritorno all'esecuzione del programma chiamante.

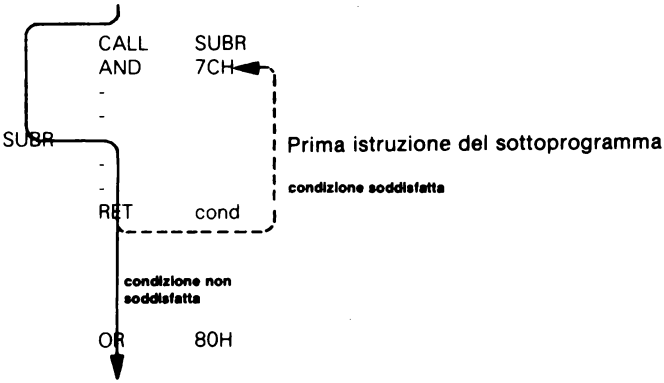
RET cond — RITORNO DA SOTTOPROGRAMMA SE LA CONDIZIONE È SODDISFATTA



		Condizione	Flag Pertinente
000	NZ	Non-Zero	Z
001	Z	Zero	Z
010	NC	Nessun Carry	C
011	C	Carry	C
100	PO	Parità dispari	P/O
101	PE	Parità pari	P/O
110	P	Segno positivo	S
111	M	Segno negativo	S

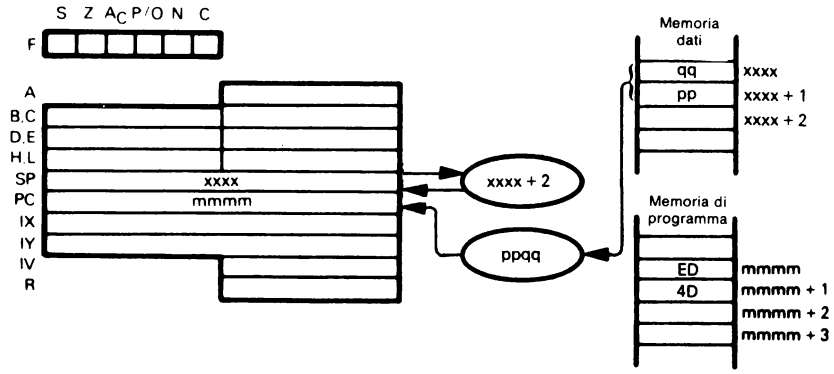
Questa istruzione è identica all'istruzione RET, tranne che non si esegue il ritorno se la condizione non è soddisfatta; altrimenti si eseguirà l'istruzione che segue in sequenza l'istruzione RET cond.

Consideriamo la sequenza di istruzioni:



Dopo l'esecuzione di RET cond, se la condizione è soddisfatta, allora l'esecuzione ritorna all'istruzione AND che segue CALL. Se la condizione non è soddisfatta, sarà eseguita l'istruzione Or, che è la successiva istruzione in sequenza.

RETI — RITORNO DALL'INTERRUZIONE

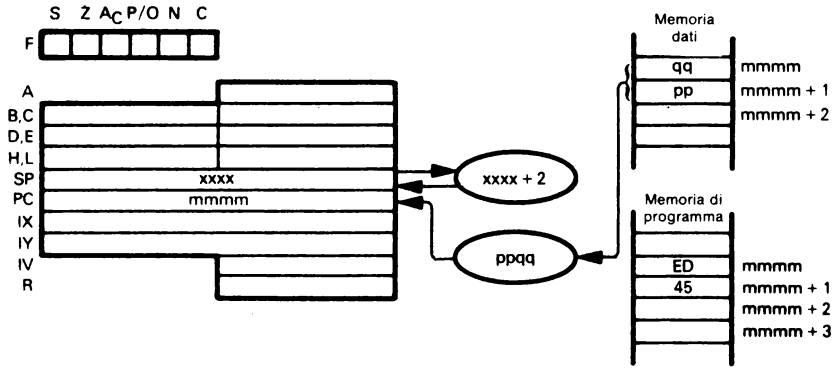


RETI
ED 4D

Sposta il contenuto dei due byte in cima allo stack nel Contatore di Programma; questi due byte forniscono l'indirizzo della prossima istruzione da eseguire. Il contenuto precedente del Contatore di Programma è perso. Incrementa lo Stack Pointer di 2 e indirizza nuovamente la cima dello stack.

Questa istruzione è usata alla fine del programma di servizio dell'interruzione e, oltre che per ridare il controllo al programma interrotto, è usata per segnalare ad un dispositivo di I/O che il programma d'interruzione è stato portato a termine. Il dispositivo di I/O deve fornire la logica necessaria per discernere il codice operativo dell'istruzione: si faccia riferimento al Capitolo 7 di An Introduction to Microcomputers: Volume II per una descrizione di come funziona l'istruzione RETI con i dispositivi della famiglia dello Z80.

RETN — RITORNO DA UNA INTERRUZIONE NON MASCHERABILE

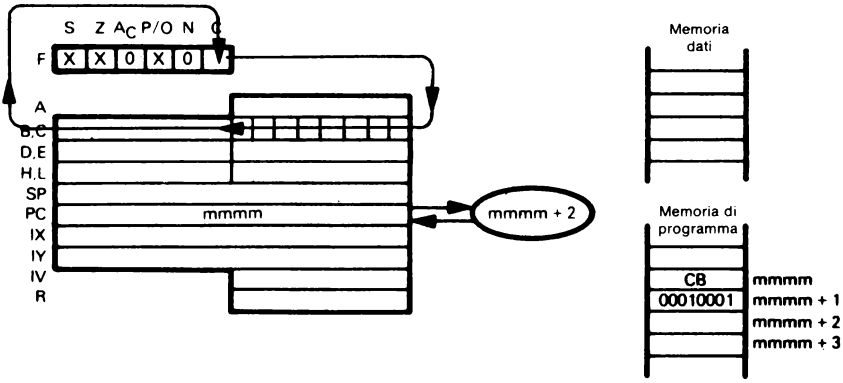


RETN
ED 45

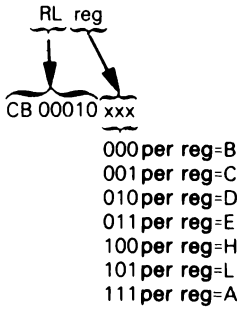
Sposta il contenuto dei due byte in cima allo stack nel Contatore di Programma; questi due byte forniscono l'indirizzo della prossima istruzione da eseguire. Il contenuto precedente del Contatore di Programma è perduto. Incrementa di 2 lo Stack Pointer per indirizzare nuovamente la cima dello stack. Ripristina la logica di abilitazione dell'interruzione per stabilire se essa avesse priorità nell'eventualità di interruzioni non mascherabili.

Questa istruzione è usata alla fine di un programma di servizio di una interruzione non mascherabile e fa sì che l'esecuzione ritorni al programma interrotto.

RL reg – RUOTA IL CONTENUTO DEL REGISTRO A SINISTRA CON CARRY



L'illustrazione mostra l'esecuzione di RL C:

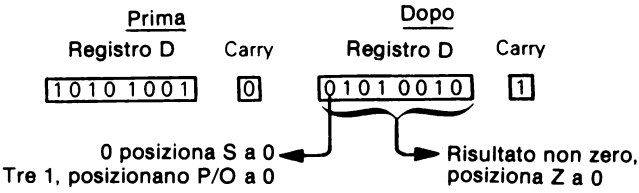


Ruota il contenuto del registro specificato a sinistra di un bit con Carry.

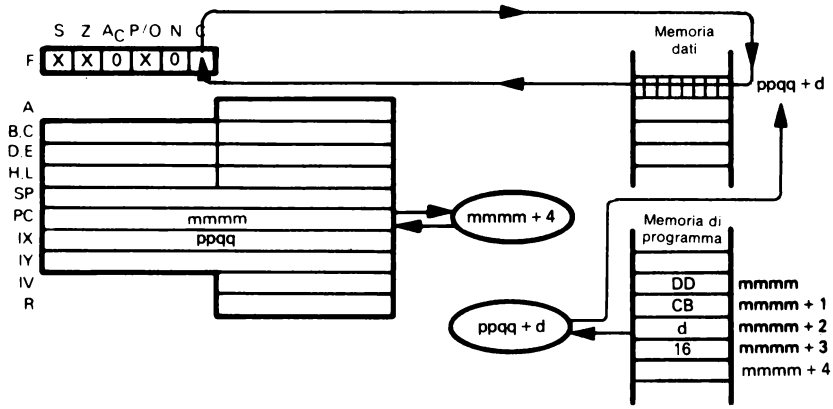
Supponiamo che D contenga $A9_{16}$ e che Carry = 0. Dopo l'esecuzione dell'istruzione

RL D

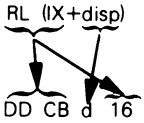
D conterrà 52_{16} e il Carry sarà uguale a 1:



RL (HL) – RUOTA IL CONTENUTO DELLA LOCAZIONE
RL (IX + disp) DI MEMORIA A SINISTRA CON CARRY
RL (IY + disp)



L'illutrazione mostra l'esecuzione di RL (IX + disp):

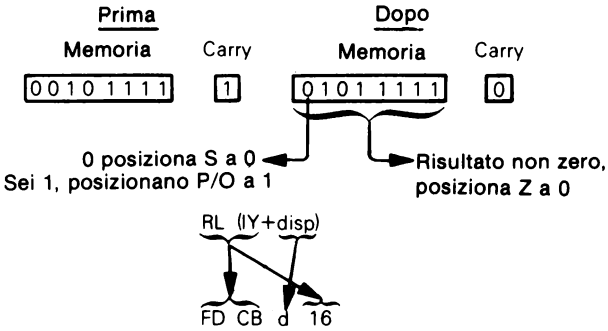


Ruota il contenuto della locazione di memoria (specificata dalla somma del contenuto del Registro Indice IX e del valore intero d del dislocamento) a sinistra di un bit con Carry.

Supponiamo che il registro IX contenga 4000_{16}, che la locazione di memoria 4007_{16} contenga $2F_{16}$ e che il Carry sia posto a 1. Dopo l'esecuzione dell'istruzione

RL (IX + 7)

la locazione di memoria 4007_{16} conterrà $5F_{16}$ e il Carry sarà 0:

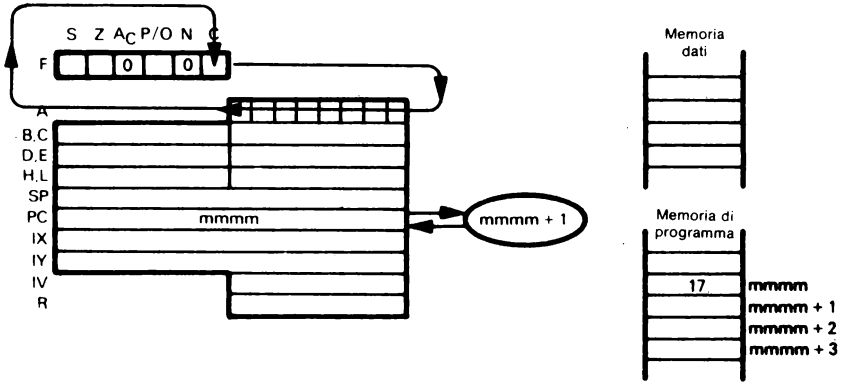


Questa istruzione è identica a RL (IX + disp), ma usa il registro IY invece del registro IX.

RL (HL)
CB 16

Ruota il contenuto della locazione di memoria (specificata dal contenuto della coppia di registri HL) a sinistra di un bit con Carry.

RLA — RUOTA L'ACCUMULATORE A SINISTRA CON CARRY



RLA
17

Ruota il contenuto dell'Accumulatore a sinistra di un bit con lo stato di Carry.

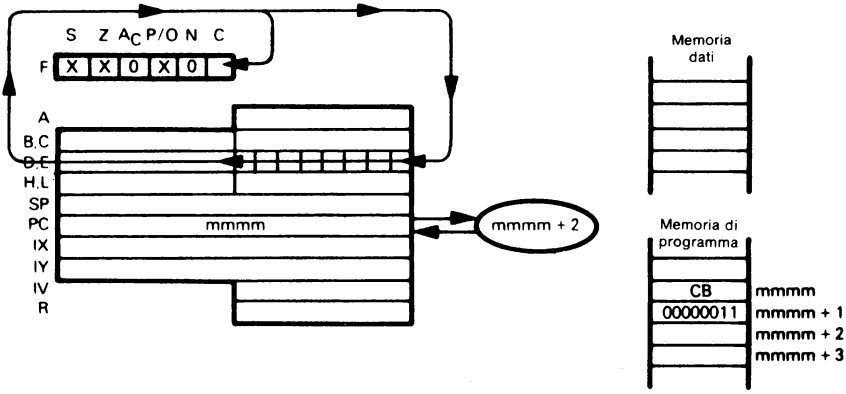
Supponiamo che l'Accumulatore contenga $2A_{16}$ e che lo stato di Carry sia posizionato ad 1. Dopo l'esecuzione dell'istruzione

RLA

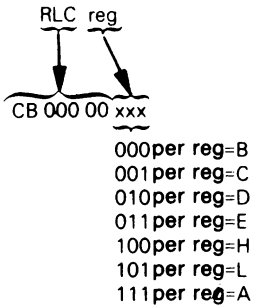
l'Accumulatore conterrà $F5_{16}$ e lo stato di Carry sarà posizionato a 0:

Prima		Dopo	
Accumulatore	Carry	Accumulatore	Carry
0111 1010	1	1111 0101	0

RLC reg — RUOTA IL CONTENUTO DEL REGISTRO A SINISTRA CON RICIRCOLO



L'illustrazione mostra l'esecuzione di RLC E:

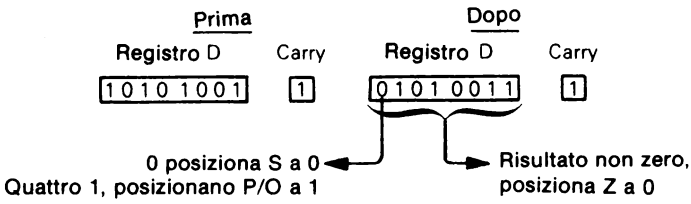


Ruota il contenuto del registro specificato a sinistra di un bit, ricopiando il bit 7 nel Carry.

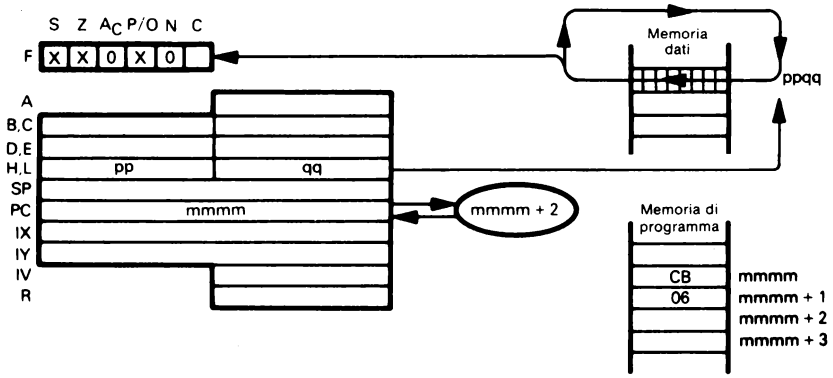
Supponiamo che il Registro D contenga $A9_{16}$ e che il Carry sia 1. Dopo l'esecuzione di

RLC D

il Registro D conterrà 53_{16} e il Carry sarà 1:



RLC (HL) – RUOTA IL CONTENUTO DELLA LOCAZIONE
RLC (IX + disp) DI MEMORIA A SINISTRA CON RICIRCOLO
RLC (IY + disp)



L'illustrazione mostra l'esecuzione di RLC (HL):

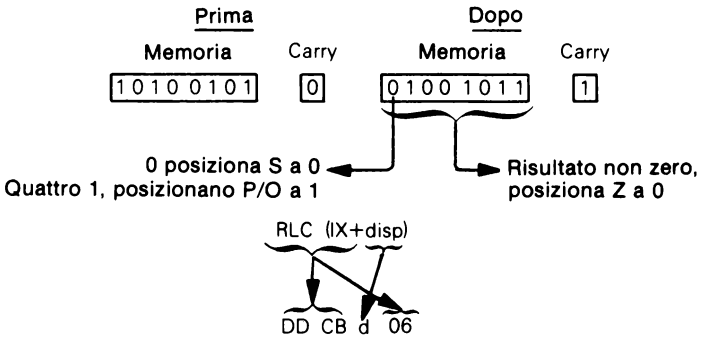
$$\underbrace{\text{RLC (HL)}}_{\text{CB 06}}$$

Ruota il contenuto della locazione di memoria (specificata dal contenuto della coppia di registri HL) a sinistra di un bit, ricopiando il bit 7 nel Carry.

Supponiamo che la coppia di registri HL contenga $54FF_{16}$. La locazione di memoria $54FF_{16}$ contenga $A5_{16}$ e il Carry sia 0. Dopo l'esecuzione di

RLC (HL)

la locazione di memoria $54FF_{16}$ conterrà $4B_{16}$ e il Carry sarà 1:

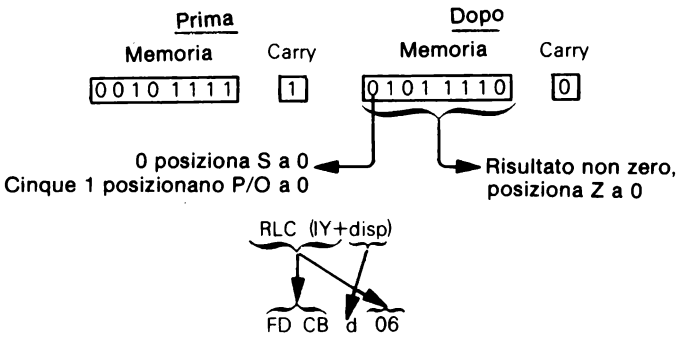


Ruota la locazione di memoria (specificata dalla somma del contenuto del registro Indice IX e del valore intero d del dislocamento) a sinistra di un bit, ricopiando il bit 7 nel Carry.

Supponiamo che il registro IX contenga 4000_{16}. Il Carry sia 1 e la locazione di memoria 4007_{16} contenga $2F_{16}$. Dopo l'esecuzione dell'istruzione

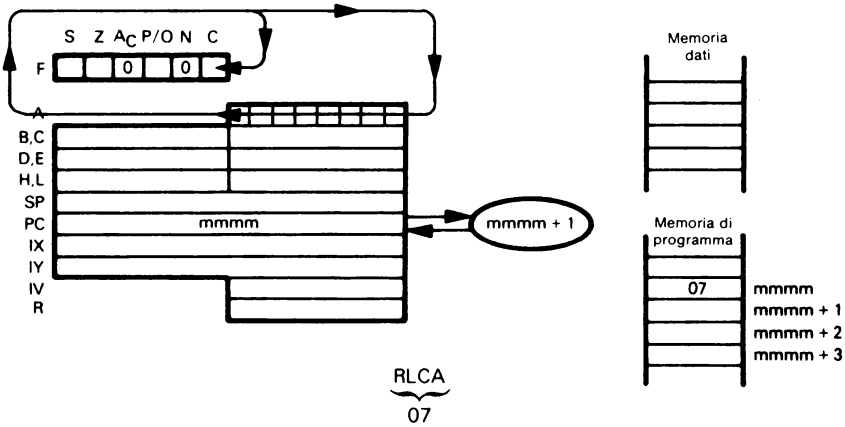
RLC (IX + 7)

la locazione di memoria 4007_{16} conterrà $5E_{16}$ e il Carry sarà 0:



Questa istruzione è identica a RLC (IX + disp), ma usa il registro IY invece del registro IX.

RLCA – RUOTA L'ACCUMULATORE A SINISTRA CON RICIRCOLO

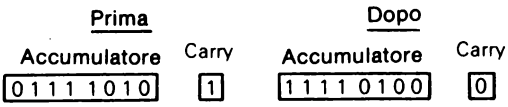


Ruota il contenuto dell'Accumulatore a sinistra di un bit, ricopiando il bit 7 nel Carry.

Supponiamo che l'Accumulatore contenga $7A_{16}$ e che lo stato del Carry sia posizionato ad 1. Dopo l'esecuzione dell'istruzione

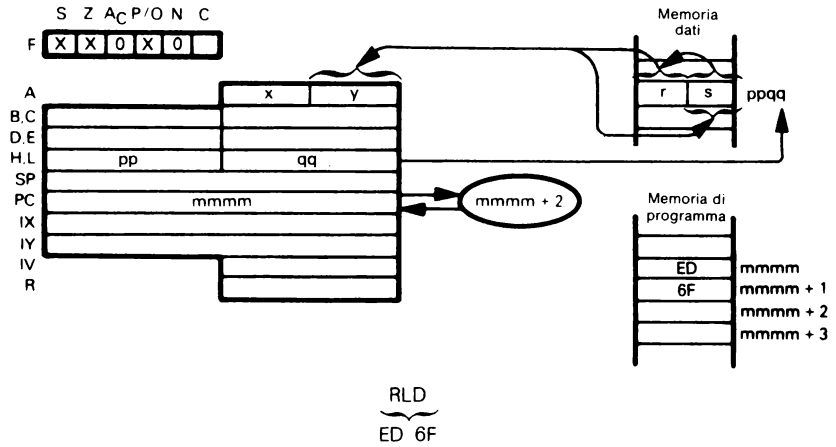
RLCA

l'Accumulatore conterrà $F4_{16}$ e lo stato del Carry sarà posizionato a 0.



RLCA dovrebbe essere usata come istruzione logica.

RLD — RUOTA UN DIGIT BCD A SINISTRA TRA L'ACCUMULATORE E UNA LOCAZIONE DI MEMORIA

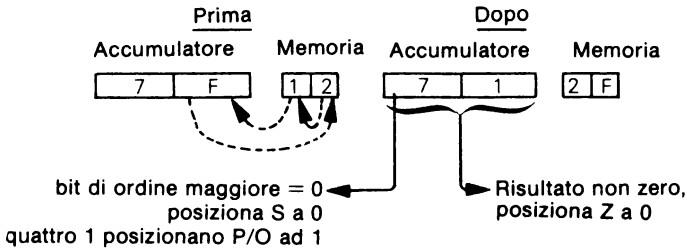


I quattro bit di ordine minore di una locazione di memoria (specificata dal contenuto della coppia di registri HL) sono ricopiati nei quattro bit di ordine maggiore della stessa locazione di memoria. Il contenuto precedente dei quattro bit di ordine maggiore sono ricopiati nei quattro bit di ordine minore dell'Accumulatore. I precedenti quattro bit di ordine minore dell'Accumulatore sono ricopiati nei quattro bit di ordine minore della locazione di memoria specificata.

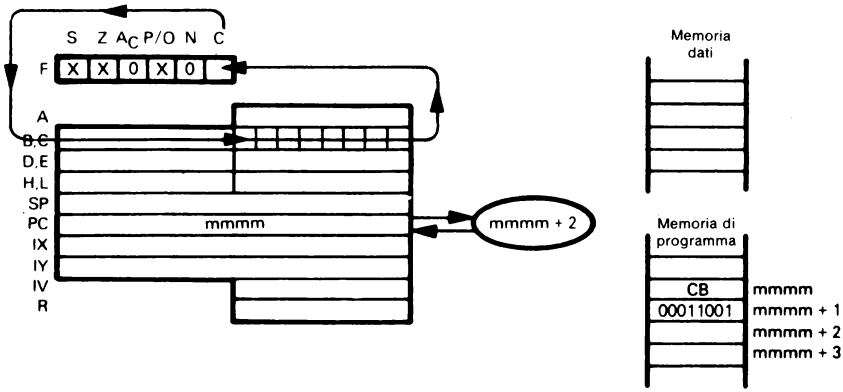
Supponiamo che l'Accumulatore contenga $7F_{16}$, che la coppia di registri HL contenga 4000_{16} e che la locazione di memoria 4000_{16} contenga 12_{16}. Dopo l'esecuzione dell'istruzione

RLD

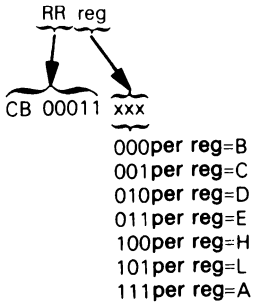
l'Accumulatore conterrà 71_{16} e la locazione di memoria 4000_{16} conterrà $2F_{16}$:



RR reg — RUOTA IL CONTENUTO DEL REGISTRO A DESTRA CON CARRY



L'illustrazione mostra l'esecuzione di RR C:

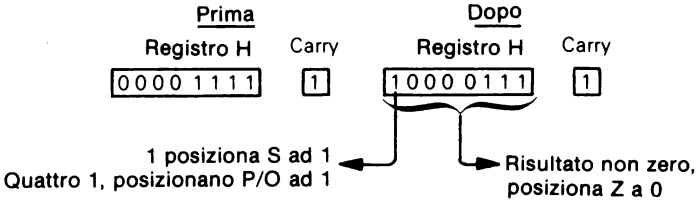


Ruota il contenuto del registro specificato a destra di un bit con Carry.

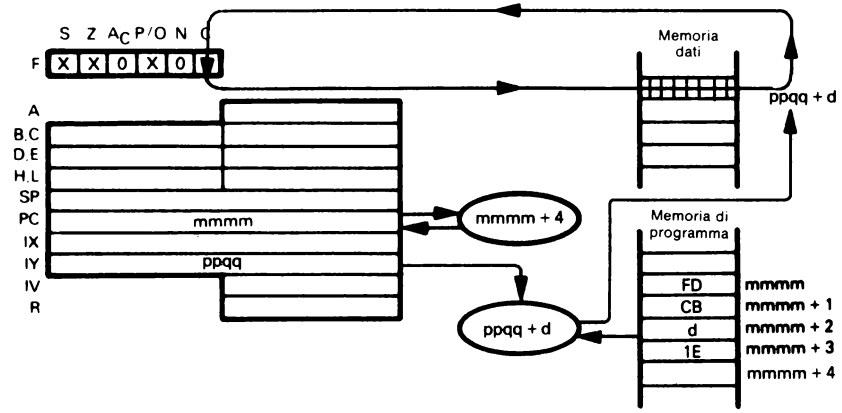
Supponiamo che il Registro H contenga $0F_{16}$ e che il Carry sia posizionato ad 1. Dopo l'esecuzione dell'istruzione

RR H

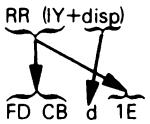
il Registro H conterrà 87_{16} e il Carry sarà 1:



RR (HL) – RUOTA IL CONTENUTO DELLA LOCAZIONE
RR (IX + disp) DI MEMORIA A DESTRA CON CARRY
RR (IY + disp)



L'illustrazione mostra l'esecuzione di RR (IY + disp):

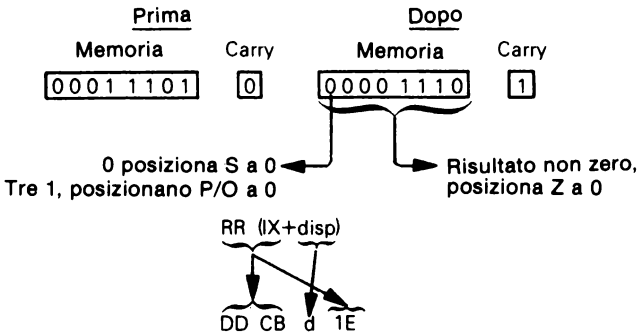


Ruota il contenuto della locazione di memoria (specificata dalla somma del contenuto del registro IY e del valore d del dislocamento) a destra di un bit con Carry.

Supponiamo che il registro IY contenga 4500_{16}; che la locazione di memoria $450F_{16}$ contenga $1D_{16}$ e che il Carry sia posizionato a 0. Dopo l'esecuzione dell'istruzione

RR (IY + 0FH)

la locazione di memoria $450F_{16}$ conterrà $0E_{16}$ e il Carry sarà 1:

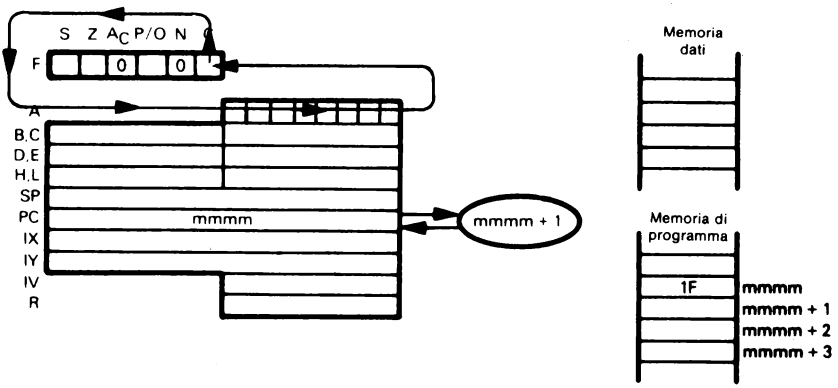


Questa istruzione è identica a RR (IY + disp), ma usa il registro IX invece del registro IY.

RR (HL)
CB 1E

Ruota il contenuto della locazione di memoria (specificata dal contenuto della coppia di registri HL) a destra di un bit con Carry.

RRA — RUOTA L'ACCUMULATORE A DESTRA CON CARRY



RRA
1F

Ruota il contenuto dell'Accumulatore a destra di un bit con lo stato di Carry.

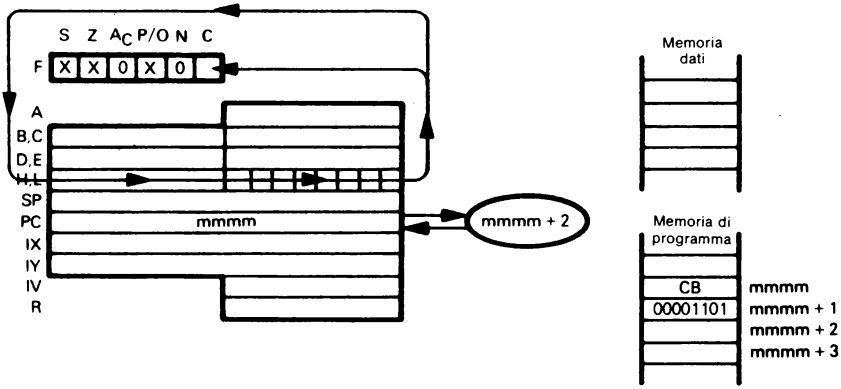
Supponiamo che l'Accumulatore contenga $7A_{16}$ e che lo stato del Carry sia posizionato ad 1. Dopo l'esecuzione dell'istruzione

RRA

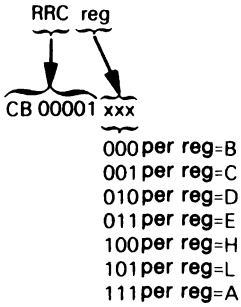
l'Accumulatore conterrà BD_{16} e lo stato del Carry sarà posizionato a 0:

Prima		Dopo	
Accumulatore	Carry	Accumulatore	Carry
0 1 1 1 1 0 1 0	1	1 0 1 1 1 1 0 1	0

RRC reg – RUOTA IL CONTENUTO DEL REGISTRO A DESTRA CON RICIRCOLO



L'illutrazione mostra l'esecuzione di RRC L:

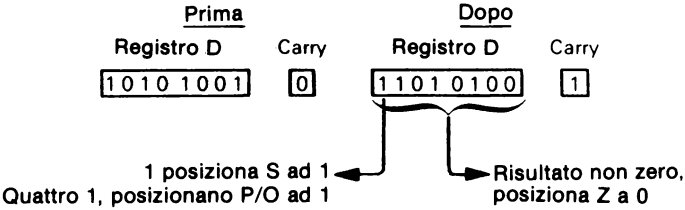


Ruota il contenuto del registro specificato a destra di un bit con ricircolo, ricopiando il bit 0 nello stato del Carry.

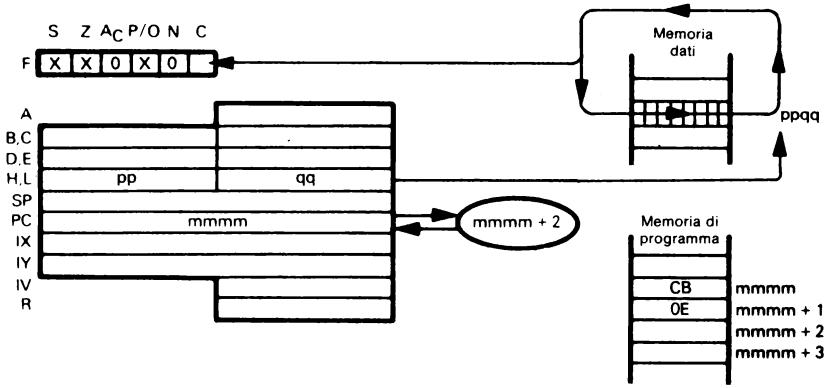
Supponiamo che il Registro D contenga $A9_{16}$ e che il Carry sia 0. Dopo l'esecuzione di

RRC D

il Registro D conterrà $D4_{16}$ e il Carry sarà 1:



RRC (HL) – RUOTA IL CONTENUTO DELLA LOCAZIONE
RRC (IX + disp) DI MEMORIA A DESTRA CON RICIRCOLO
RRC (IY + disp)



L'illustrazione mostra l'esecuzione di RRC (HL):

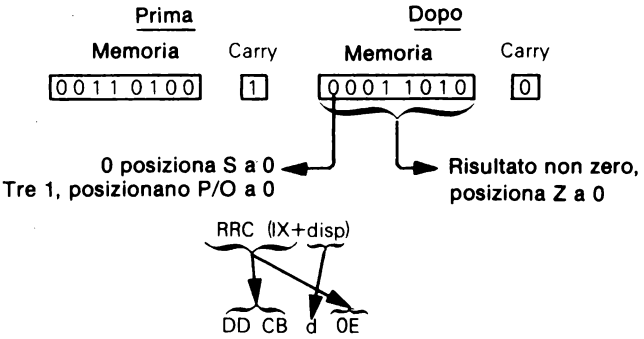
$$\underbrace{\text{RRC (HL)}}_{\text{CB 0E}}$$

Ruota il contenuto della locazione di memoria (specificata dal contenuto della coppia di registri HL) a destra di un bit con ricircolo, ricopiando il bit 0 nello stato del Carry.

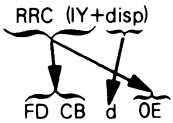
Supponiamo che la coppia di registri HL contenga 4500_{16}, che la locazione di memoria 4500_{16} contenga 34_{16} e che il Carry sia posizionato ad 1. Dopo l'esecuzione di

RRC (HL)

la locazione di memoria 4500_{16} conterrà $1A_{16}$ e il Carry sarà 0:

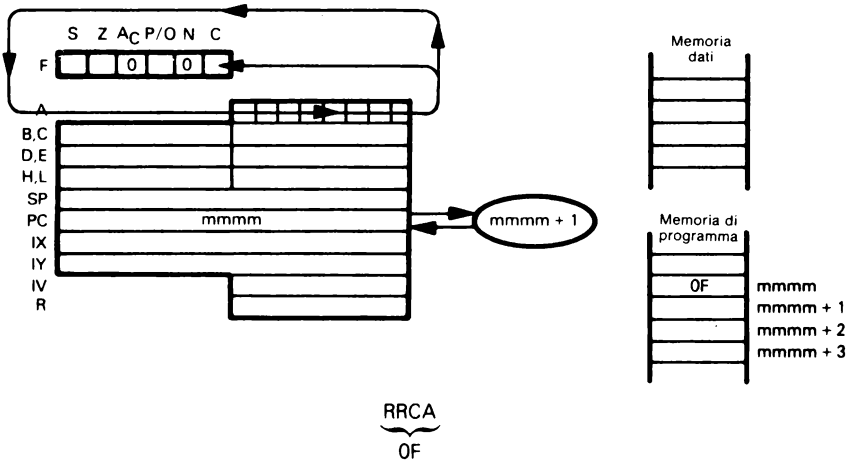


Ruota il contenuto della locazione di memoria (specificata dalla somma del contenuto del registro IX e del valore d del dislocamento) a destra di un bit con ricircolo, ricopiando il bit 0 nello stato del Carry.



Questa istruzione è identica all'istruzione RRC (IX + disp), ma usa il registro IY invece del registro IX.

RRCA — RUOTA L'ACCUMULATORE A DESTRA CON RICIRCOLO



Ruota il contenuto dell'Accumulatore a destra di un bit con ricircolo, ricopiando il bit 0 nello stato del Carry.

Supponiamo che l'Accumulatore contenga $7A_{16}$ e che lo stato del Carry sia posizionato ad 1. Dopo l'esecuzione dell'istruzione

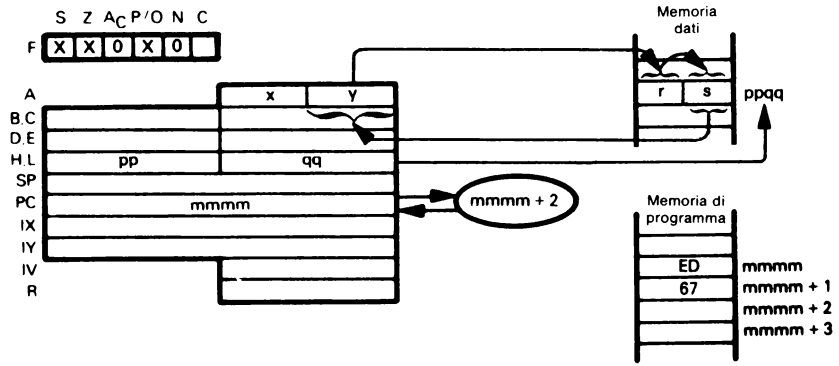
RRCA

l'Accumulatore contiene $3D_{16}$ e lo stato del Carry sarà posizionato a 0:

Prima		Dopo	
Accumulatore	Carry	Accumulatore	Carry
0111 1010	1	0011 1101	0

RRCA dovrebbe essere usata come un'istruzione logica.

RRD — RUOTA UN DIGIT BCD A DESTRA TRA L'ACCUMULATORE E LA LOCAZIONE DI MEMORIA



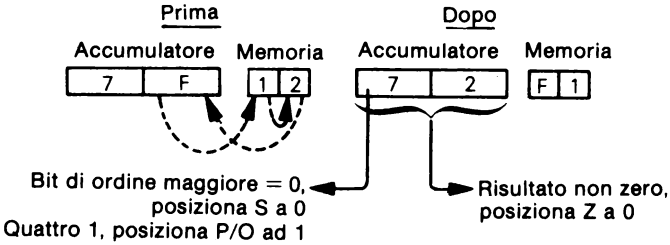
RRD
ED 67

I quattro bit di ordine maggiore della locazione di memoria (specificata dal contenuto della coppia di registri HL) sono ricopiati nei quattro bit di ordine minore della stessa locazione di memoria. Il contenuto precedente dei quattro bit di peso minore è ricopiato nei quattro bit di peso minore dell'Accumulatore. I precedenti quattro bit di ordine minore dell'Accumulatore sono ricopiati nei quattro bit di ordine maggiore della locazione di memoria specificata.

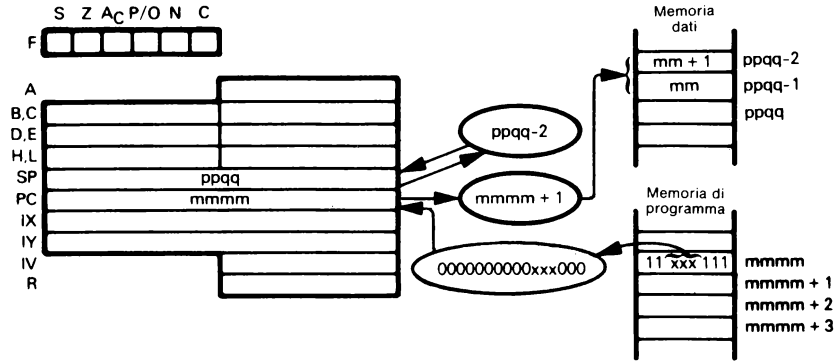
Supponiamo che l'Accumulatore contenga $7F_{16}$, che la coppia di registri HL contenga 4000_{16} e che la locazione di memoria 4000_{16} contenga 12_{16}. Dopo l'esecuzione della istruzione

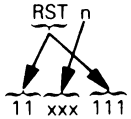
RRD

l'Accumulatore conterrà 72_{16} e la locazione di memoria 4000_{16} conterrà $F1_{16}$:



RST n – RIAVVIO (RESTART)





Chiama il sottoprogramma originato all'indirizzo basso di memoria specificato da n.

Quando viene eseguita l'istruzione

RST 18H

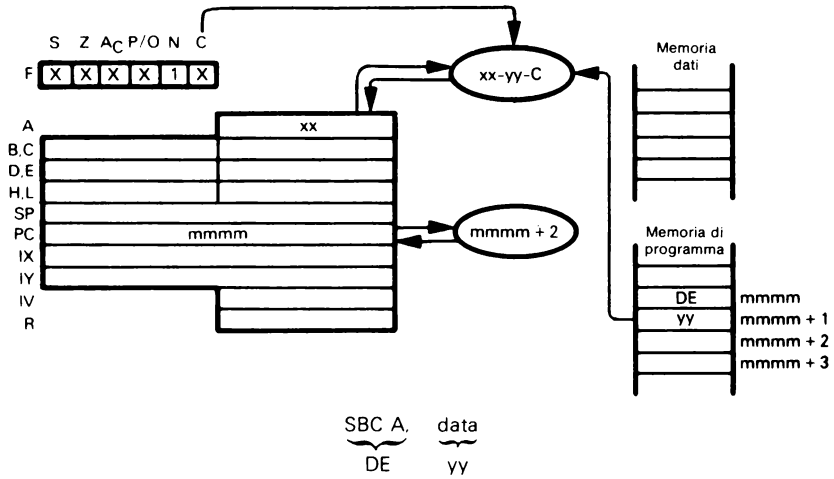
si chiama il sottoprogramma originato alla locazione di memoria 0018_{16}. Il contenuto precedente del Contatore di Programma è spinto in cima allo stack.

Usualmente, l'istruzione RST è usata congiuntamente ad un processo di interruzione, come descritto nel Capitolo 5.

CHIAMATA DI UN SOTTOPROGRAMMA USANDO RST

Se il vostro programma non usa tutti i codici dell'istruzione RST per servire le interruzioni, non trascurate la possibilità di chiamare sottoprogrammi che usano istruzioni RST. Si ponga l'origine di sottoprogrammi frequentemente usati ad indirizzi appropriati di RST; questi sottoprogrammi possono essere chiamati con un'istruzione a singolo byte RST invece che con un'istruzione CALL a tre byte.

SBC A,data — SOTTRAZIONE IMMEDIATA DI UN DATO DALL'ACCUMULATORE CON PRESTITO

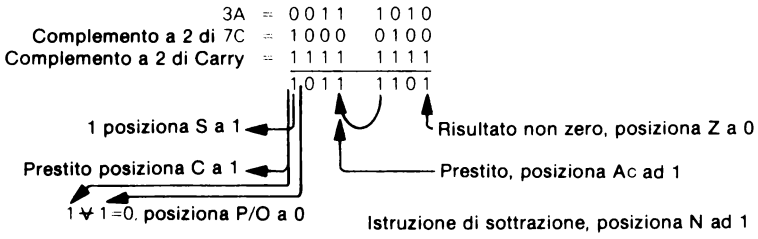


Sottrae il contenuto del secondo byte di codice oggetto e lo stato del Carry dall'Accumulatore.

Supponiamo che xx = $3A_{16}$ e che il Carry = 1. Dopo l'esecuzione dell'istruzione

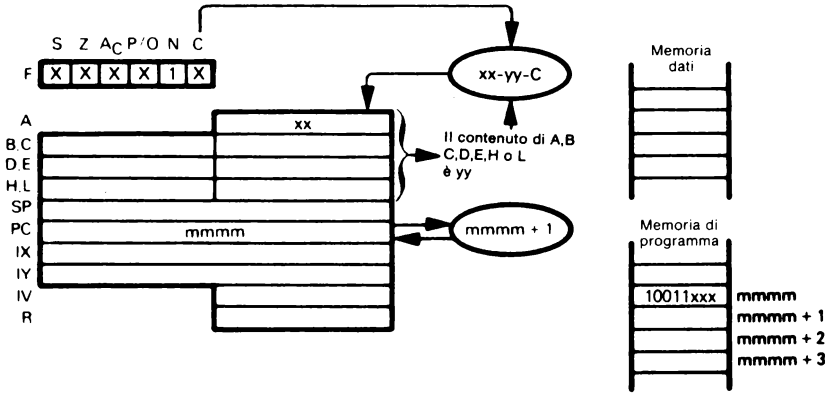
SBC A,7CH

l'Accumulatore conterrà BD_{16}.



È da notare che il carry risultante è complementato.

SBC A,reg – SOTTRAE IL REGISTRO CON PRESTITO DALL'ACCUMULATORE



SBC A, reg
10011 xxx

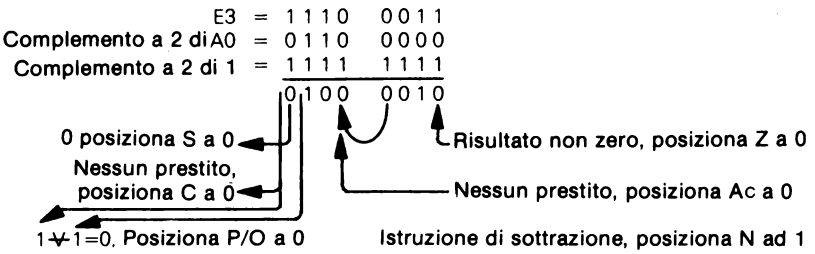
000 per reg=B
001 per reg=C
010 per reg=D
011 per reg=E
100 per reg=H
101 per reg=L
111 per reg=A

Sottrae il contenuto del registro specificato e lo stato del Carry dall'Accumulatore.

Supponiamo che xx = $E3_{16}$, che il Registro E contenga $A0_{16}$ e che il Carry = 1. Dopo l'esecuzione dell'istruzione

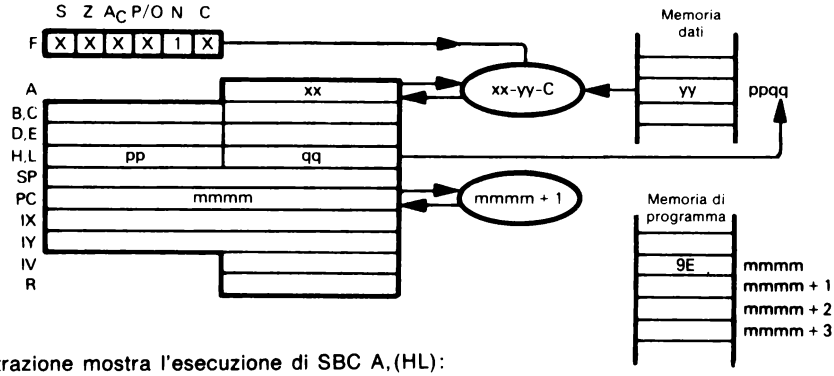
SBC A,E

l'Accumulatore conterrà 42_{16}.



È da notare che il carry risultante è complementato.

SBC A,(HL) – SOTTRAE LA MEMORIA ED IL CARRY
SBC A,(IX + disp) DALL'ACCUMULATORE
SBC A,(IY + disp)



L'illutrazione mostra l'esecuzione di SBC A,(HL):

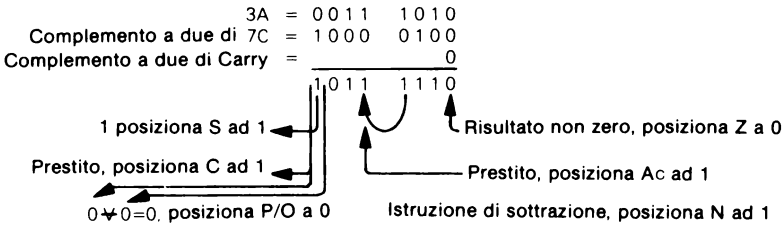
$$\underbrace{\text{SBC A,(HL)}}_{\text{9E}}$$

Sottrae il contenuto della locazione di memoria (specificata dal contenuto della coppia di registri HL) e il Carry dall'Accumulatore.

Supponiamo che il Carry = 0, ppqq = 4000_{16}, xx = $3A_{16}$ e che la locazione di memoria 4000_{16} contenga $7C_{16}$. Dopo l'esecuzione dell'istruzione

SBC A,(HL)

l'Accumulatore conterrà BE_{16}.



È da notare che il carry risultante è complementato.

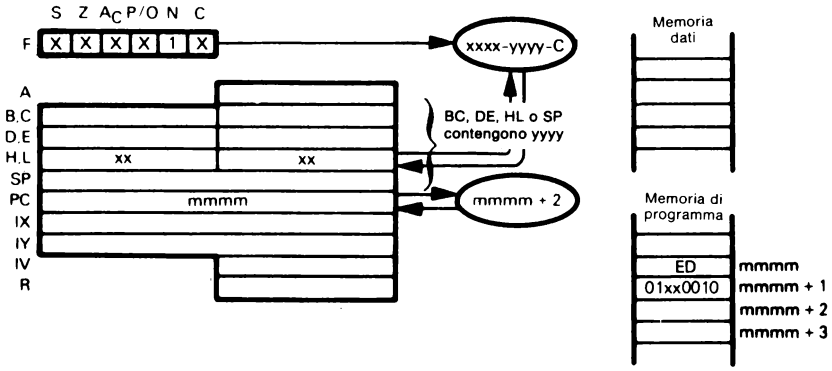
$$\underbrace{\text{SBC A,(IX}}_{\text{DD 9E}}\underbrace{\text{+disp)}}_{\text{d}}$$

Sottrae il contenuto della locazione di memoria (specificata dalla somma del contenuto del registro IX e del valore d del dislocamento) e il Carry dall'Accumulatore.

$$\underbrace{\text{SBC A,(IY}}_{\text{FD 9E}}\underbrace{\text{+disp)}}_{\text{d}}$$

Questa istruzione è identica a SBC A,(IX + disp), tranne che essa usa il registro IY invece del registro IX.

SBC HL,rp – SOTTRAE LA COPPIA DI REGISTRI CON CARRY DA H ED L



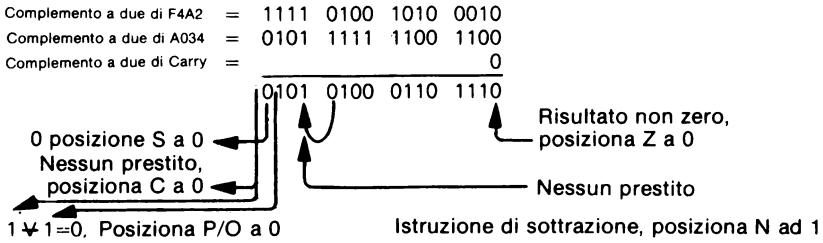
00 per rp è la coppia di registri BC
01 per rp è la coppia di registri DE
10 per rp è la coppia di registri HL
11 per rp è lo Stack Pointer

Sottrae il contenuto della coppia di registri indicata e lo stato del Carry dalla coppia di registri HL.

Supponiamo che HL contenga $F4A2_{16}$, che BC contenga $A034_{16}$ e che Carry = 0. Dopo l'esecuzione dell'istruzione

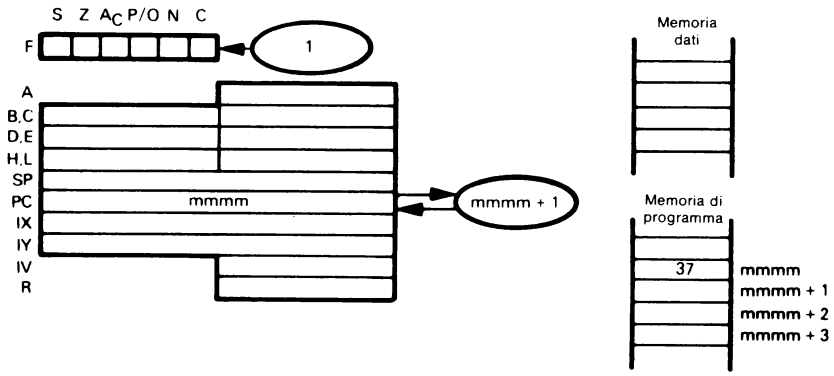
SBC HL,BC

la coppia di registri HL conterrà $546E_{16}$:



È da notare che il carry risultante è complementato.

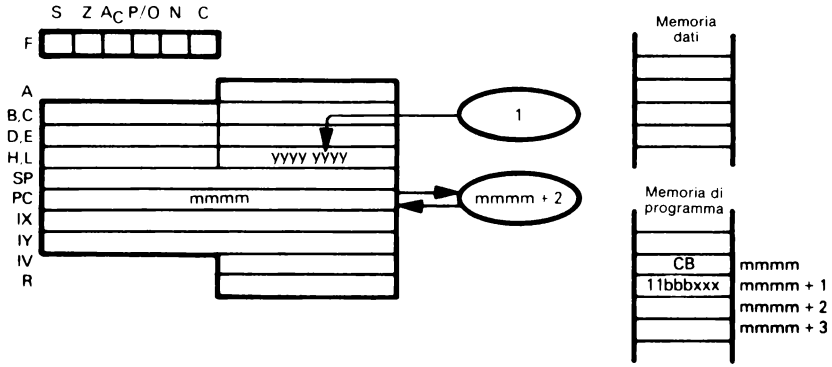
SCF – POSIZIONA IL FLAG DI CARRY

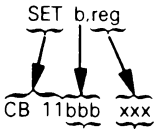


SCF
37

Quando viene eseguita l'istruzione SCF, si posiziona a 1 lo stato del Carry, senza riguardo al suo valore precedente. Non si influenza il contenuto di nessun altro stato o registro.

SET b,reg — POSIZIONA IL BIT INDICATO DEL REGISTRO





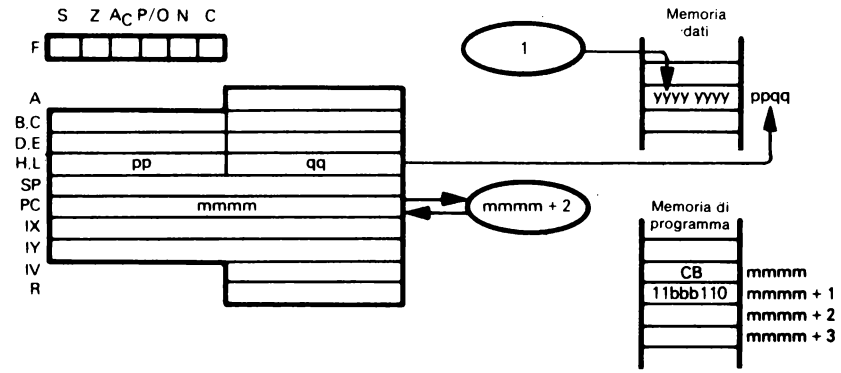
Bit	bbb	xxx	Registro
0	000	000	B
1	001	001	C
2	010	010	D
3	011	011	E
4	100	100	H
5	101	101	L
6	110	111	A
7	111		

Il bit indicato da SET nel registro specificato viene posizionato ad 1. Dopo l'esecuzione dell'istruzione

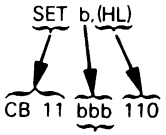
SET 2,L

Il bit 2 del Registro L sarà posizionato ad 1. (Il bit 0 è il bit meno significativo).

SET b,(HL) – POSIZIONA IL BIT b DELLA POSIZIONE
SET b,(IX + disp) DI MEMORIA INDICATA
SET b,(IY + disp)



L'illustrazione mostra l'esecuzione di SET b(HL). Il bit 0 è il bit meno significativo.



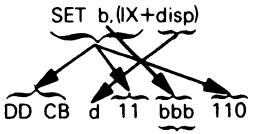
Bit posizionato	bbb
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Posiziona ad 1 il bit indicato nella locazione di memoria indicata da HL.

Supponiamo che HL contenga 4000_{16}. Dopo l'esecuzione dell'istruzione

SET 5,(HL)

il bit 5 della posizione di memoria 4000_{16} sarà 1.



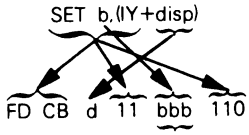
bbb è lo stesso di SET b,(HL)

Posiziona ad 1 il bit indicato nella locazione di memoria indicata dalla somma del Registro Indice IX e del dislocamento.

Supponiamo che il Registro Indice IX contenga 4000_{16}. Dopo l'esecuzione di

SET 6,(IX + 5H)

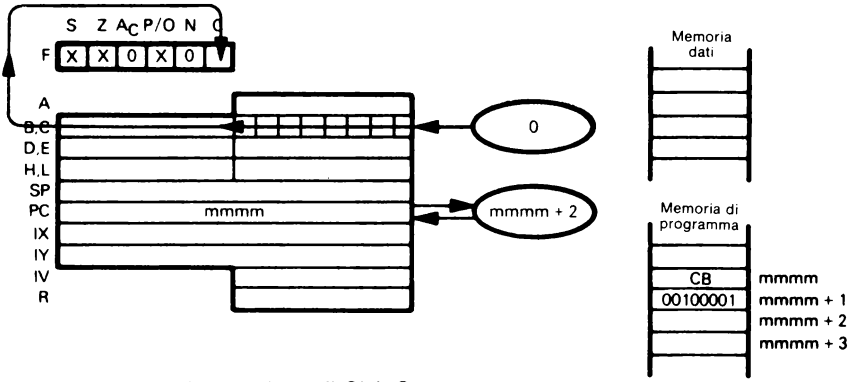
il bit 6 della locazione di memoria 4005_{16} sarà 1.



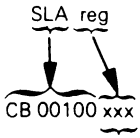
bbb è lo stesso di SET b,(HL)

Questa istruzione è identica a SET b,(IX + disp), tranne che essa usa il registro IY invece del Registro IX.

SLA reg — SPOSTA IL CONTENUTO DEL REGISTRO A SINISTRA IN MODO ARITMETICO



L'illustrazione mostra l'esecuzione di SLA C:



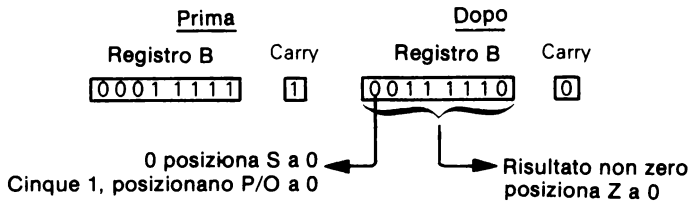
000 per reg=B
001 per reg=C
010 per reg=D
011 per reg=E
100 per reg=H
101 per reg=L
111 per reg=A

Sposta il contenuto del registro specificato a sinistra di un bit, posizionando a 0 il bit meno significativo.

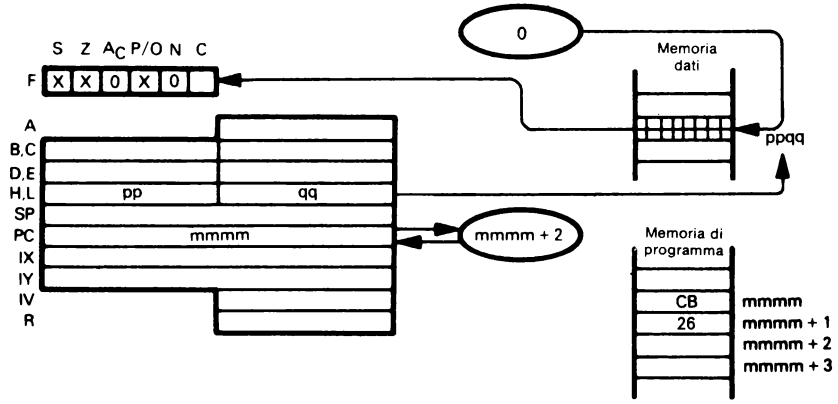
Supponiamo che il Registro B contenga $1F_{16}$ e che il Carry = 1. Dopo l'esecuzione di

SLA B

il Registro B conterrà $3E_{16}$ e il Carry sarà 0.



SLA (HL) – SPOSTA IL CONTENUTO DELLA LOCAZIONE SLA (IX + disp) DI MEMORIA A SINISTRA IN MODO ARITMETICO SLA (IY + disp)



L'illustrazione mostra l'esecuzione di SLA (HL):

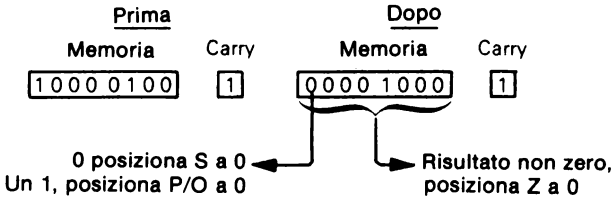
$$\underbrace{\text{SLA (HL)}}_{\text{CB 26}}$$

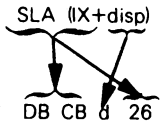
Sposta il contenuto della locazione di memoria (specificata dal contenuto della coppia di registri HL) a sinistra di un bit, posizionando a 0 il bit meno significativo.

Supponiamo che la coppia di registri HL contenga 4500_{16}, che la locazione di memoria 4500_{16} contenga 84_{16} e che il Carry = 0. Dopo l'esecuzione di

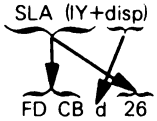
SLA (HL)

la locazione di memoria 4500_{16} conterrà 08_{16} e il Carry sarà 1.



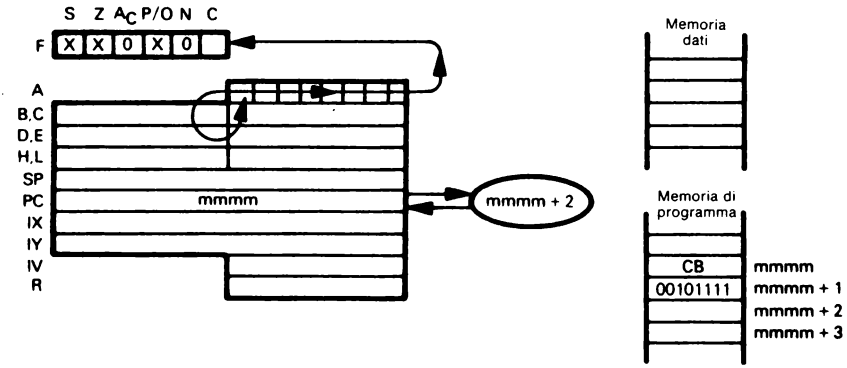


Sposta il contenuto della locazione di memoria (specificata dalla somma del contenuto del registro IX e del valore d del dislocamento) a sinistra di un bit in modo aritmetico, posizionando a 0 il bit meno significativo).

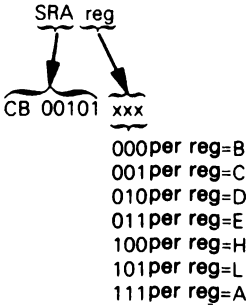


Questa istruzione è identica a SLA (IX + disp), ma usa il registro IY invece del registro IX.

SRA reg — SPOSTA IL CONTENUTO DEL REGISTRO A DESTRA IN MODO ARITMETICO



L'illustrazione mostra l'esecuzione di SRA A:

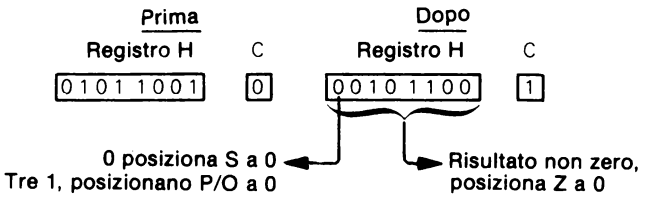


Sposta il registro specificato a destra di un bit. Il bit più significativo rimane immutato.

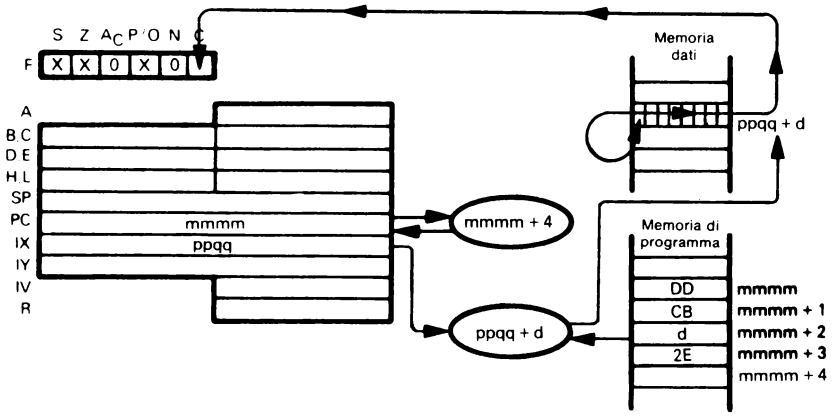
Supponiamo che il Registro H contenga 59_{16} e che Carry = 0. Dopo l'esecuzione della istruzione

SRA H

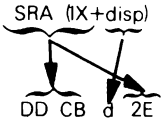
il registro H conterrà $2C_{16}$ e il Carry sarà 1.



SRA (HL) – SPOSTA A DESTRA IL CONTENUTO DELLA SRA (IX + disp) POSIZIONE DI MEMORIA IN MODO ARITMETICO SRA (IY + disp)



L'illustrazione mostra l'esecuzione di SRA (IX + disp):

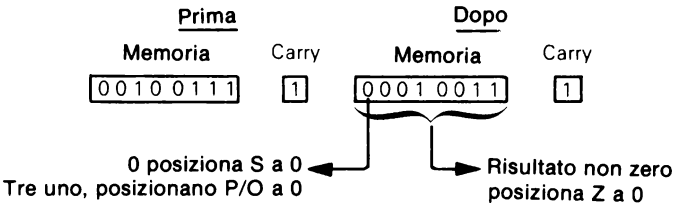


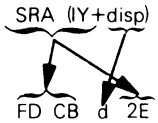
Sposta il contenuto della locazione di memoria (specificata dalla somma del contenuto del Registro IX e del valore d del dislocamento) a destra. Il bit più significativo rimane immutato.

Supponiamo che il Registro IX contenga 3400_{16}, che la locazione di memoria $34AA_{16}$ contenga 27_{16} e che Carry = 1. Dopo l'esecuzione di

SRA (IX + 0AAH)

la locazione di memoria $34AA_{16}$ conterrà 13_{16} e il Carry sarà 1.

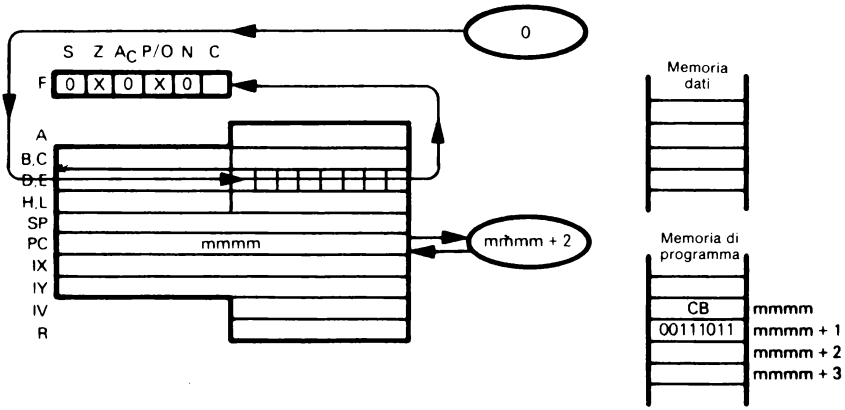




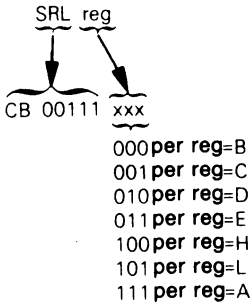
Questa istruzione è identica a SRA (IX + disp), ma usa il registro IY invece del registro IX.

Sposta il contenuto della locazione di memoria (specificata dal contenuto della coppia di registri HL) a destra di un bit. Il bit più significativo rimane immutato.

SRL reg — SPOSTA IL CONTENUTO DEL REGISTRO A DESTRA IN MODO LOGICO



L'illustrazione mostra l'esecuzione di SRL E

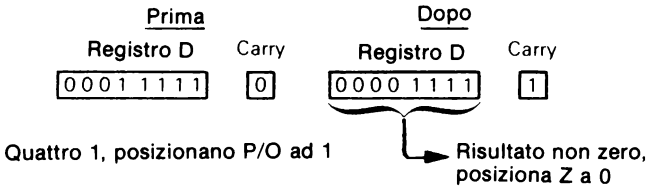


Sposta il contenuto del registro specificato a destra di un bit. Il bit più significativo è posizionato a 0.

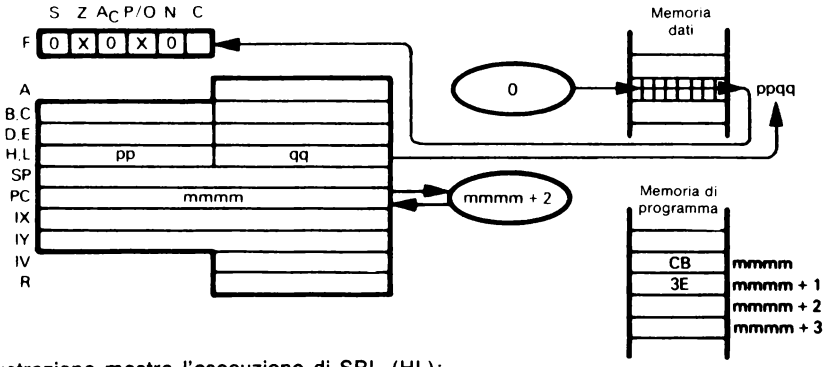
Supponiamo che il Registro D contenga $1F_{16}$ e che Carry = 0. Dopo l'esecuzione di

SRL D

il Registro D conterrà $0F_{16}$ e il Carry sarà 1.



SRL (HL) – SPOSTA IL CONTENUTO DELLA LOCAZIONE SRL (IX + disp) DI MEMORIA A DESTRA IN MODO LOGICO SRL (IY + disp)



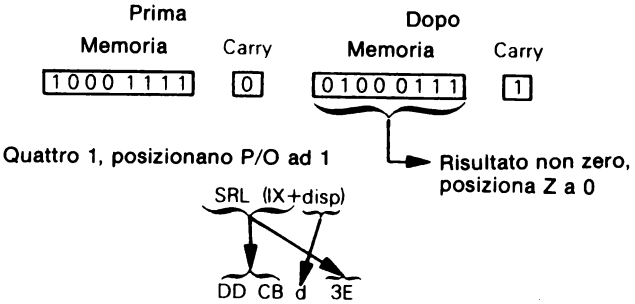
L'illustrazione mostra l'esecuzione di SRL (HL):

Sposta il contenuto della locazione di memoria (specificata dal contenuto della coppia di registri HL) a destra di un bit. Il bit più significativo è posizionato a 0.

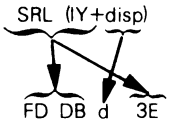
Supponiamo che la coppia di registri HL contenga 2000_{16}, che la locazione di memoria 2000_{16} contenga $8F_{16}$ e che Carry = 0. Dopo l'esecuzione di

SRL (HL)

la locazione di memoria 2000_{16} conterrà 47_{16} e il Carry sarà 1.

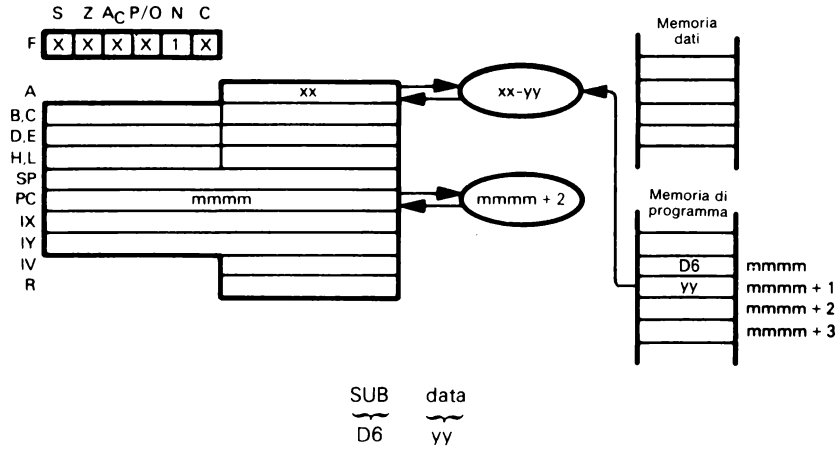


Sposta il contenuto della locazione di memoria (specificata dalla somma del contenuto del registro IX e del valore d del dislocamento) a destra di un bit. Il bit più significativo è posizionato a 0.



Questa istruzione è identica a SRL (IX + disp), ma usa il registro IY invece del registro IX.

SUB data — SOTTRAZIONE IMMEDIATA DALL'ACCUMULATORE

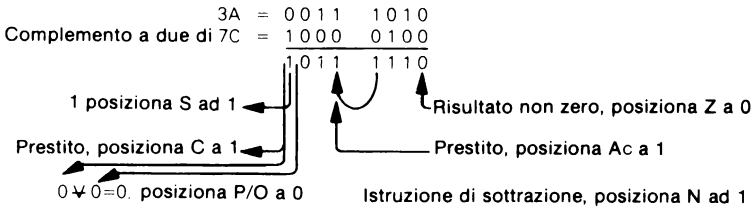


Sottrae il contenuto del secondo byte del codice oggetto dall'Accumulatore.

Supponiamo che xx = $3A_{16}$. Dopo l'esecuzione dell'istruzione

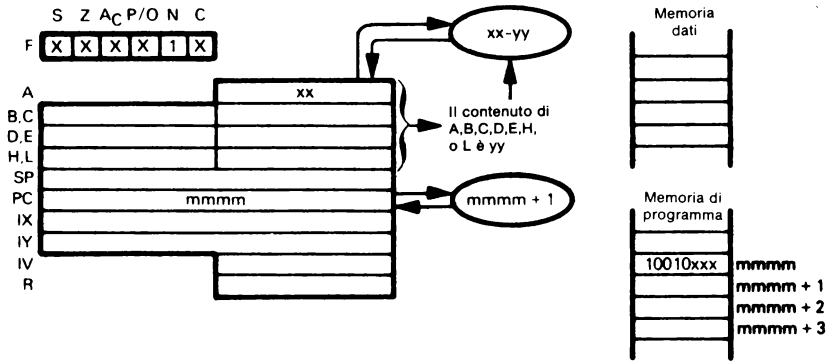
SUB 7CH

l'Accumulatore conterrà BE_{16}.



È da notare che il carry risultante è complementato.

SUB reg — SOTTRAE IL REGISTRO DALL'ACCUMULATORE



SUB reg

10010 xxx

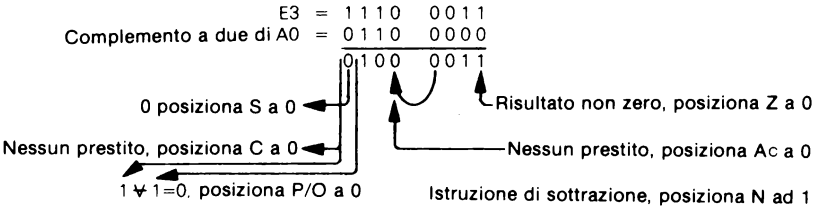
000 per reg=B
001 per reg=C
010 per reg=D
011 per reg=E
100 per reg=H
101 per reg=L
111 per reg=A

Sottrae il contenuto del registro specificato dall'Accumulatore.

Supponiamo che xx = E3 e che il Registro H contenga $A0_{16}$. Dopo l'esecuzione di

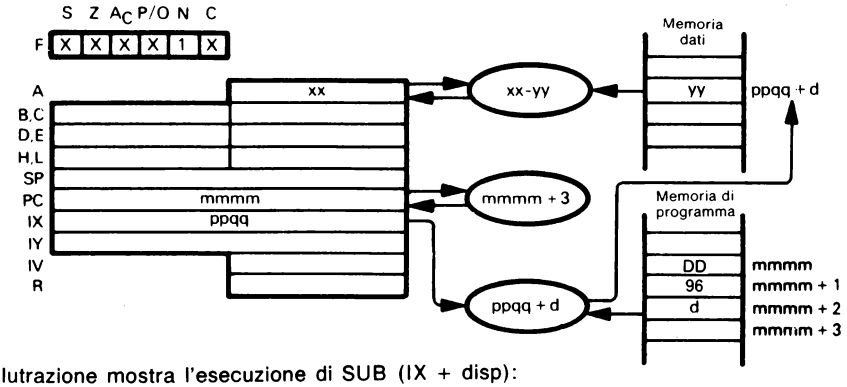
SUB H

l'Accumulatore conterrà 43_{16}.



È da notare che il carry risultante è complementato.

SUB (HL) – SOTTRAE LA MEMORIA DALL'ACCUMULATORE
SUB (IX + disp)
SUB (IY + disp)



L'illutrazione mostra l'esecuzione di SUB (IX + disp):

$$\underbrace{\text{SUB (IX}}\underbrace{\text{+disp)}}$$

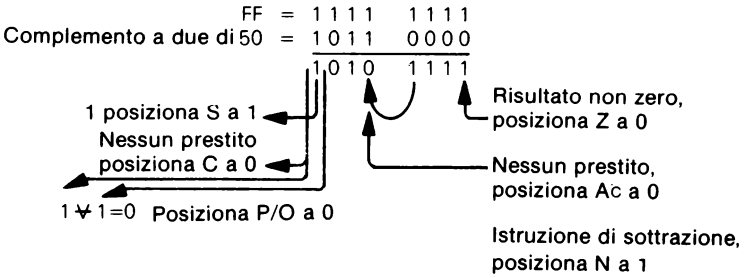
DD 96 d

Sottrae il contenuto della locazione di memoria (specificata dalla somma del contenuto del registro IX e del valore d del dislocamento) dall'Accumulatore.

Supponiamo che ppqq = 4000_{16}, xx = FF_{16} e che la locazione di memoria $40FF_{16}$ contenga 50_{16}. Dopo l'esecuzione di

SUB (IX + 0FFH)

l'Accumulatore conterrà AF_{16}.



È da notare che il carry risultante è complementato.

$$\underbrace{\text{SUB (IY}}\underbrace{\text{+disp)}}$$

FD 96 d

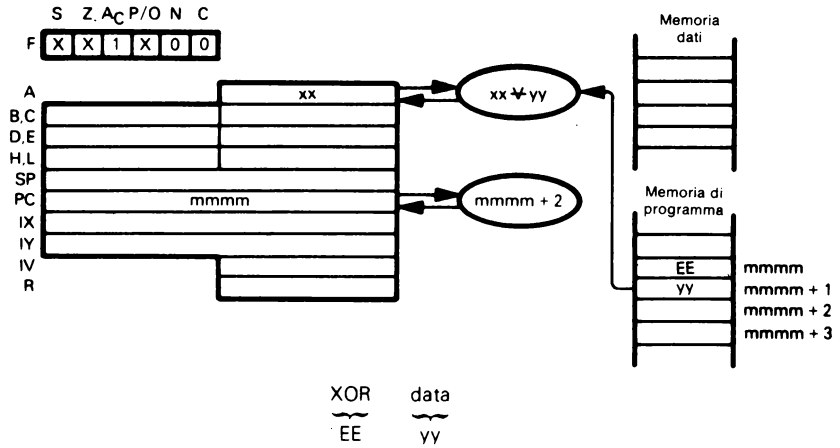
Questa istruzione è identica a SUB (IX + disp), tranne che essa usa il registro IY invece del registro IX.

$$\underbrace{\text{SUB (HL)}}$$

96

Sottrae il contenuto della locazione di memoria (specificata dal contenuto della coppia di registri HL), dall'Accumulatore.

XOR data — OR ESCLUSIVO IMMEDIATO CON L'ACCUMULATORE



Fa l'OR esclusivo del contenuto del secondo byte del codice oggetto con l'Accumulatore.

Supponiamo che xx = $3A_{16}$. Dopo l'esecuzione dell'istruzione

XOR 7CH

l'Accumulatore conterrà 46_{16}.

```
3A = 0011 1010
7C = 0111 1100
     ---------
     0100 0110
```

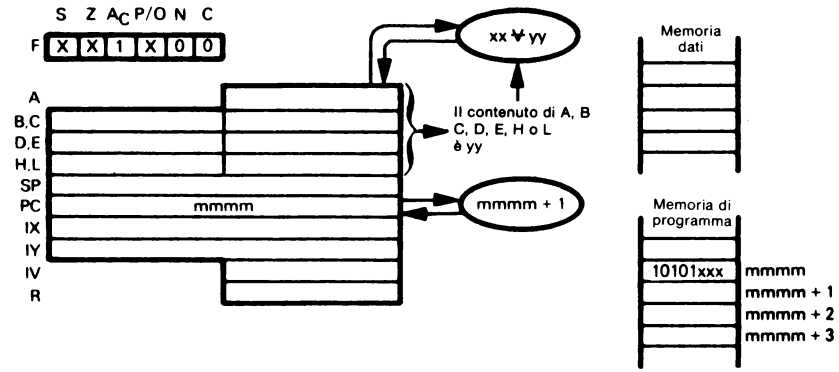
0 posiziona S a 0

Risultato non zero, posiziona Z a 0

Tre bit ad 1, posizionano P/O a 0

L'istruzione OR esclusivo è usata per provare cambiamenti negli stati dei bit.

XOR reg — OR ESCLUSIVO DEL REGISTRO CON L'ACCUMULATORE



XOR reg

10101 xxx

000 per reg=B
001 per reg=C
010 per reg=D
011 per reg=E
100 per reg=H
101 per reg=L
111 per reg=A

Fa l'OR esclusivo del contenuto del registro specificato con l'Accumulatore.

Supponiamo che xx = $E3_{16}$ e che il Registro E contenga $A0_{16}$. Dopo l'esecuzione della istruzione

XOR E

l'Accumulatore conterrà 43_{16}.

```
E3 = 1110 0011
A0 = 1010 0000
     ---------
     0100 0011
```

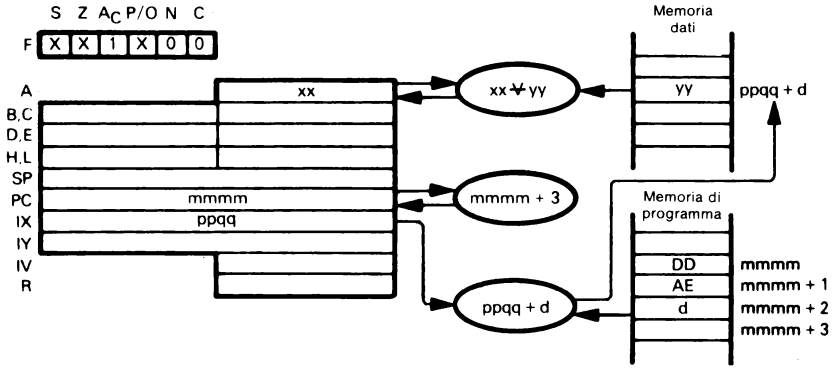
0 posiziona S a 0

Risultato non zero, posiziona Z a 0

Tre bit ad 1, posizionano P/O a 0

L'istruzione OR esclusivo è usata per verificare cambiamenti negli stati dei bit.

XOR (HL) – OR ESCLUSIVO DELLA MEMORIA CON
XOR (IX + disp) L'ACCUMULATORE
XOR (IY + disp)



L'illustrazione mostra l'esecuzione di XOR (IX + disp):

$$\underbrace{\text{XOR (IX}}_{\text{DD AE}}\underbrace{\text{+disp)}}_{\text{d}}$$

Fa l'OR esclusivo del contenuto della locazione di memoria (specificata dalla somma del contenuto del registro IX e del valore d del dislocamento) con l'Accumulatore.

Supponiamo che xx = $E3_{16}$, ppqq = 4500_{16} e che la locazione di memoria $45FF_{16}$ contenga $A0_{16}$. Dopo l'esecuzione dell'istruzione

XOR (IX + 0FFH)

l'Accumulatore conterrà 43_{16}.

```
E3 = 1110  0011
A0 = 1010  0000
     0100  0011
```

0 posiziona S a 0

Risultato non zero, posiziona Z a 0

Tre bit ad 1, posizionano P/O a 0

$$\underbrace{\text{XOR (IY}}_{\text{FD AE}}\underbrace{\text{+disp)}}_{\text{d}}$$

Questa istruzione è identica a XOR (IX + disp), tranne che essa usa il registro IY invece del registro IX.

$$\underbrace{\text{XOR (HL)}}_{\text{AE}}$$

Fa l'OR esclusivo del contenuto della locazione di memoria (specificata dal contenuto della coppia di registri HL) con l'Accumulatore.

COMPATIBILITÀ 8080A/Z80

CARATTERISTICHE DI COMPATIBILITÀ 8080A/Z80

Sebbene il microprocessore Z80 possa essere certamente usato per i suoi stessi meriti, una delle sue caratteristiche importanti è la compatibilità con il microprocessore 8080A. Questa compatibilità ha le seguenti caratteristiche:

1) Tutte le istruzioni in linguaggio macchina dello 8080A sono pure istruzioni in linguaggio macchina dello Z80.
2) Tutti i registri dell'8080A sono pure registri dello Z80 (vedi la Tabella 3-6).
3) Quasi tutti i programmi per 8080A verranno eseguiti su uno Z80, con qualche diversità di poca importanza che sarà indicata più avanti.
4) Lo Z80 ha istruzioni, registri e altre caratteristiche non presenti sull'8080A, cosicchè programmi per Z80 generalmente non saranno eseguiti su processori 8080A.

CONVERSIONE DI LIVELLO ASSEMBLY 8080A/Z80

Si noti che questa compatibilità non si estende a statement di programma sorgente in linguaggio assembly, poiché gli assemblatori Z80 e 8080A usano differenti codici operativi mnemonici. **La Tabella 3-7 contiene un elenco dei codici mnemonici dell'8080A e i corrispondenti codici dello Z80, mentre la Tabella 3-8 è il medesimo elenco organizzato secondo i codici Z80.**

CODICI OPERATIVI NON UTILIZZATI DELL'8080A

I lettori noteranno le limitazioni dei codici binari che questa compatibilità pone sulle caratteristiche supplementari del microelaboratore Z80. L'8080A ha alcuni codici operativi non utilizzati (vedi la Tabella 3-9) che sono usati per alcune istruzioni supplementari dello Z80. Ma non esiste un numero sufficiente di tali codici da coprire il grande numero di caratteristiche in una forma semplice.

CODICI OPERATIVI A 2 PAROLE

Quindi, molte delle istruzioni Z80 aggiunte richiedono codici operativi a 2 parole. La prima parola è CB, DD, ED o FD. Si noti il seguente significato di questi codici dalla Tabella 3-9:

CB — Operazione su un registro o su un bit
DD — Operazione implicante il registro IX
ED — Un'istruzione miscellanea non di 8080A non riscontrabile altrove.
FD — Operazione implicante il registro IY

La seconda parola del codice operativo descrive la reale operazione da compiere.

ESECUZIONE DELLE ISTRUZIONI PIÙ VELOCEMENTE E PIÙ LENTAMENTE

Il risultato finale è che queste istruzioni a più parole si eseguono abbastanza lentamente (e usano più memorie) poiché si richiede un accesso in memoria in più. Il lettore dovrebbe considerare questa variazione nei tempi di esecuzione e cercare di usare istruzioni più veloci da eseguire dove possibile. Questo avvertimento si applica in particolare modo alle istruzioni in più di scorrimento (RLC, RRC, RL, RR, SRA, SRL) e alle istruzioni implicanti i registri indice IX e IY.

INCOMPATIBILITÀ 8080A/Z80

Esistono alcune incompatibilità di poco conto tra 8080A e Z80. Eccole:

1) Lo Z80 usa il flag P (o P/O) per indicare overflow di complemento a due dopo operazioni aritmetiche. L'8080A usa sempre questo flag per la parità.
2) Lo Z80 e l'8080A eseguono l'istruzione DAA in modo differente. Sullo Z80 questa istruzione correggerà una sottrazione decimale come una addizione decimale. Sull'8080A, essa correggerà solo l'addizione decimale.

3) Le istruzioni di rotazione dello Z80 azzerano il flag Ac. Le istruzioni di rotazione dell'8080A non influenzano il flag Ac.

Tabella 3-6. Corrispondenze tra i Registri ed i Flag dello Z80 e dell'8080A

Registro dello Z80	Registro dell'8080A
A	A
A'	Nessuno
B	B
B'	Nessuno
C	C
C'	Nessuno
D	D
D'	Nessuno
E	E
E'	Nessuno
F	Metà Meno Significativa di PSW
F'	Nessuno
H	H
H'	Nessuno
I	Nessuno
IX	Nessuno
IY	Nessuno
L	L
L'	Nessuno
R	Nessuno
PC	PC
SP	SP
Coppie di Registri dello Z80	**Coppie di Registri dell'8080A**
BC	B
DE	D
HL	H
AF	PSW
Flag dello Z80	**Flag dell'8080A**
C (Carry)	C (Carry)
H (Half-Carry)	AC (Carry Ausiliario)
N (Subtract)	Nessuno
P/O (Parità/Overflow)	P (Parità)
S (Segno)	S (Segno)
Z (Zero)	Z (Zero)

INCOMPATIBILITÀ 8085/Z80

Lo Z80 non è compatibile con le caratteristiche supplementari del microcomputer 8085. I codici usati per RIM e SIM sull'8085 sono usati per salti relativi (NZ e NC) sullo Z80.

INCOMPATIBILITÀ DI TEMPORIZZAZIONE

Le temporizzazioni delle istruzioni sull'8080A, sull'8085 e sullo Z80 sono tutte differenti. I programmi che dipendono da precise temporizzazioni mediante istruzioni verranno eseguiti in maniera appropriata solo sull'elaboratore per il quale essi sono stati scritti.

Il flag N sullo Z80 occupa il bit 2 del registro F; il corrispondente bit nella Parola di Stato dell'Elaboratore dell'8080A è sempre ad «1» logico.

Tabella 3-7. Corrispondenza tra i codici Mnemonici dell'8080A e dello Z80

Codici mnemonici dell'8080A		Codici mnemonici dello Z80	
ACI	data	ADC	A,data
ADC	reg or M	ADC	A,reg or (HL)
ADD	reg or M	ADD	A,reg or (HL)
ADI	data	ADD	A,data
ANA	reg or M	AND	reg or (HL)
ANI	data	AND	data
CALL	addr	CALL	addr
CC	addr	CALL	C,addr
CM	addr	CALL	M,addr
CMA		CPL	
CMC		CCF	
CMP	reg or M	CP	reg or (HL)
CNC	addr	CALL	NC,addr
CNZ	addr	CALL	NZ,addr
CP	addr	CALL	P,addr
CPE	addr	CALL	PE,addr
CPI	data	CP	data
CPO	addr	CALL	PO,addr
CZ	addr	CALL	Z,addr
DAA		DAA	
DAD	rp	ADD	HL,rp
DCR	reg or M	DEC	reg or (HL)
DCX	rp	DEC	rp
DI		DI	
EI		EI	
HLT		HALT	
IN	port	IN	A,(port)
INR	reg or M	INC	reg or (HL)
INX	rp	INC	rp
JC	addr	JP	C,addr
JM	addr	JP	M,addr
JMP	addr	JP	addr
JNC	addr	JP	NC,addr
JP	addr	JP	P,addr
JNZ	addr	JP	NZ,addr
JPE	addr	JP	PE,addr
JPO	addr	JP	PO,addr
JZ	addr	JP	Z,addr
LDA	addr	LD	A,(addr)
LDAX	B or D	LD	A,(BC) or (DE)

Codici mnemonici dell'8080A		Codici mnemonici dello Z80	
LHLD	addr	LD	HL,(addr)
LXI	rp,data16	LD	rp,data16
MOV	reg,reg or M	LD	reg,reg or (HL)
MOV	reg or M,reg	LD	reg or (HL),reg
MVI	reg or M,data	LD	reg or (HL),data
NOP		NOP	
ORA	reg or M	OR	reg or (HL)
ORI	data	OR	data
OUT	port	OUT	(port),A
PCHL		JP	(HL)
POP	pr	POP	pr
PUSH	pr	PUSH	pr
RAL		RLA	
RAR		RRA	
RC		RET	C
RET		RET	
RLC		RLCA	
RM		RET	M
RNC		RET	NC
RNZ		RET	NZ
RP		RET	P
RPE		RET	PE
RPO		RET	PO
RRC		RRCA	
RST	n	RST	n
RZ		RET	Z
SBB	reg or M	SBC	A,reg or (HL)
SBI	data	SBC	A,data
SHLD	addr	LD	(addr),HL
SPHL		LD	SP,HL
STA	addr	LD	(addr),A
STAX	B or D	LD	(BC) or (DE),A
STC		SCF	
SUB	reg or M	SUB	reg or (HL)
SUI	data	SUB	data
XCHG		EX	DE,HL
XRA	reg or M	XOR	reg or (HL)
XRI	data	XOR	data
XTHL		EX	(SP),HL

Tabella 3-8. Corrispondenza tra i codici Mnemonici dello Z80 e dell'8080A

Codici mnemonici dello Z80		Codici mnemonici dell'8080A	
ADC	A,data	ACI	data
ADC	A,(HL)	ADC	M
ADC	A,reg	ADC	reg
ADC	A,(xy + disp)	—	
ADC	HL,rp	—	
ADD	A,data	ADI	data
ADD	A,(HL)	ADD	M
ADD	A,reg	ADD	reg
ADD	A,(xy + disp)	—	
ADD	HL,rp	DAD	rp
ADD	IX,pp	—	
ADD	IY,rr	—	
AND	data	ANI	data
AND	(HL)	ANA	M
AND	reg	ANA	reg
AND	(xy + disp)	—	
BIT	b,(HL)	—	
BIT	b,reg	—	
BIT	b,(xy + disp)	—	
CALL	addr	CALL	addr
CALL	C,addr	CC	addr
CALL	M,addr	CM	addr
CALL	NC,addr	CNC	addr
CALL	NZ,addr	CNZ	addr
CALL	P,addr	CP	addr
CALL	PE,addr	CPE	addr
CALL	PO,addr	CPO	addr
CALL	Z,addr	CZ	addr
CCF		CMC	
CP	data	CPI	data
CP	(HL)	CMP	M
CP	reg	CMP	reg
CP	(xy + disp)	—	
CPD		—	
CPDR		—	
CPI		—	
CPIR		—	
CPL		CMA	
DAA		DAA	
DEC	(HL)	DCR	M
DEC	reg	DCR	reg
DEC	rp	DCX	rp
DEC	xy	—	
DEC	(xy + disp)	—	
DI		DI	
DJNZ	disp	—	
EI		EI	
EX	AF,AF'	—	
EX	DE,HL	XCHG	
EX	(SP),HL	XTHL	
EX	(SP),xy	—	
EXX		—	
HALT		HLT	
IM	m	—	
IN	A,(port)	IN	port
IN	reg,(C)	—	
INC	(HL)	INR	M
INC	reg	INR	reg

Codici mnemonici dello Z80		Codici mnemonici dell'8080A	
INC	rp	INX	rp
INC	xy	—	
INC	(xy + disp)	—	
IND		—	
INDR		—	
INI		—	
INIR		—	
JP	addr	JMP	addr
JP	C,addr	JC	addr
JP	(HL)	PCHL	
JP	M,addr	JM	addr
JP	NC,addr	JNC	addr
JP	NZ,addr	JNZ	addr
JP	P,addr	JP	addr
JP	PE,addr	JPE	addr
JP	PO,addr	JPO	addr
JP	Z,addr	JZ	addr
JP	xy	—	
JR	C,disp	—	
JR	disp	—	
JR	NC,disp	—	
JR	NZ,disp	—	
JR	Z,disp	—	
LD	A,(addr)	LDA	addr
LD	A,(BC) or (DE)	LDAX	B or D
LD	A,I	—	
LD	A,R	—	
LD	(addr),A	STA	addr
LD	(addr),BC or DE	—	
LD	(addr),HL	SHLD	addr
LD	(addr),SP	—	
LD	(addr),xy	—	
LD	(BC) or (DE),A	STAX	B or D
LD	BC or DE,(addr)	—	
LD	HL,(addr)	LHLD	addr
LD	(HL),data	MVI	M,data
LD	(HL),reg	MOV	M,reg
LD	I,A	—	
LD	R,A	—	
LD	reg,data	MVI	reg,data
LD	reg,(HL)	MOV	reg,M
LD	reg,reg	MOV	reg,reg
LD	reg,(xy + disp)	—	
LD	rp,data16	LXI	rp,data16
LD	SP,(addr)	—	
LD	SP,HL	SPHL	
LD	SP,xy	—	
LD	xy,data16	—	
LD	xy,(addr)	—	
LD	(xy + disp),data	—	
LD	(xy + disp),reg	—	
LDD		—	
LDDR		—	
LDI		—	
LDIR		—	
NEG		—	
NOP		NOP	
OR	data	ORI	data

— indica che non c'è alcuna istruzione corrispondente.

Tabella 3-8. Corrispondenza tra i codici Mnemonici dello Z80 e dell'8080A (segue)

Codici mnemonici dello Z80		Codici mnemonici dell'8080A	
OR	(HL)	ORA	M
OR	reg	ORA	reg
OR	(xy + disp)	—	
OTDR		—	
OTIR		—	
OUT	(C),reg	—	
OUT	(port),A	OUT	port
OUTD		—	
OUTI		—	
POP	pr	POP	pr
POP	xy	—	
PUSH	pr	PUSH	pr
PUSH	xy	—	
RES	b,(HL)	—	
RES	b,reg	—	
RES	b,(xy + disp)	—	
RET		RET	
RET	C	RC	
RET	M	RM	
RET	NC	RNC	
RET	NZ	RNZ	
RET	P	RP	
RET	PE	RPE	
RET	PO	RPO	
RET	Z	RZ	
RETI		—	
RETN		—	
RL	(HL)	—	
RL	reg	—	
RL	(xy + disp)	—	
RLA		RAL	
RLC	(HL)	—	
RLC	reg	—	
RLC	(xy + disp)	—	
RLCA		RLC	
RLD		—	

Codici mnemonici dello Z80		Codici mnemonici dell'8080A	
RR	(HL)	—	
RR	reg	—	
RR	(xy + disp)	—	
RRA		RAR	
RRC	(HL)	—	
RRC	reg	—	
RRC	(xy + disp)	—	
RRCA		RRC	
RRD		—	
RST	n	RST	n
SBC	A,data	SBI	data
SBC	A,(HL)	SBB	M
SBC	A,reg	SBB	reg
SBC	A,(xy + disp)	—	
SBC	HL,rp	—	
SCF		STC	
SET	b,(HL)	—	
SET	b,reg	—	
SET	b,(xy + disp)	—	
SLA	(HL)	—	
SLA	reg	—	
SLA	(xy + disp)	—	
SRA	(HL)	—	
SRA	reg	—	
SRA	(xy + disp)	—	
SRL	(HL)	—	
SRL	reg	—	
SRL	(xy + disp)	—	
SUB	data	SUI	data
SUB	(HL)	SUB	M
SUB	reg	SUB	reg
SUB	(xy + disp)	—	
XOR	data	XRI	data
XOR	(HL)	XRA	M
XOR	reg	XRA	reg
XOR	(xy + disp)	—	

— indica che non c'è alcuna istruzione corrispondente.

Tabella 3-9. Codici operativi inutilizzati dell'8080A
e loro significato per lo Z80

Codice operativo dell'8080A	Uso di Z80
08	EX AF,AF'
10	DJNZ disp
18	JR disp
20 (RIM on 8085)	JR NZ,disp
28	JR Z,disp
30 (SIM on 8085)	JR NC,disp
38	JR C,disp
CB	BIT, RES, RL, RLC, RR, RRC, SET, SLA, SRA, SRL
D9	EXX
DD	Tutte le istruzioni coinvolgono il Registro IX.

ED	ADC HL,rp
	CPD
	CPDR
	CPI
	CPIR
	IM m
	IN reg,(C)
	IND
	INDR
	INI
	INIR
	LD A,I
	LD A,R
	LD (addr),rp
	LD I,A
	LD R,A
	LD rp,(addr)
	LDD
	LDDR
	LDI
	LDIR
	NEG
	OTDR
	OTIR
	OUT (C),reg
	OUTD
	OUTI
	RETI
	RETN
	RLD
	RRD
	SBC HL,rp

FD	Tutte le istruzioni coinvolgono il Registro IY.

CONVENZIONI DELL'ASSEMBLATORE Z80 DELLA ZILOG

L'assemblatore standard Z80 è disponibile da produttori di Z80 e dalle reti in time-sharing più importanti: fa pure parte di gran parte dei sistemi di sviluppo. Sono disponibili versioni di cross-assemblatori per gran parte dei grandi computer e per molti minicomputer.

STRUTTURA DI CAMPO DELL'ASSEMBLATORE

Le istruzioni in linguaggio assembly hanno la struttura di campo standard (vedi la Tabella 2-1). I delimitatori richiesti sono:

1) **Due punti dopo una label, tranne che per le pseudo-operazioni EQU, DEFL e MACRO, che richiedono uno spazio.**
2) **Uno spazio dopo il codice operativo.**
3) **Una virgola tra operandi nel camo operandi. (Ricordatelo)**
4) **Un punto e virgola prima di un commento.**
5) **Parentesi tonde per riferimenti in memoria.**

Tipiche istruzioni in linguaggio assembly Z80 sono:

```
START:  LD      A,(1000)  ;PRENDI LA LUNGHEZZA
        ADD     HL,DE
        HALT
```

LABEL

L'assemblatore permette label di sei caratteri; il primo carattere deve essere una lettera, mentre i caratteri successivi devono essere letteri, numeri, ?, o il carattere di sottolinea ( ). Useremo solo lettere maiuscole o numeri, anche se alcune versioni dell'assemblatore permettono lettere minuscole ed altri simboli.

NOMI RISERVATI

Alcuni nomi sono riservati come parole chiave e non dovrebbero essere usati dal programmatore. Essi sono i nomi dei registri (A, B, C, D, E, H, L, I, R), i nomi dei registri doppi (IX, IY, SP), i nomi dei registri (AF, BC, DE, HL, AF', BC', DE', HL') e gli stati dei quattro flag controllabili (C, NC, Z, NZ, M, P, PE, PO).

PSEUDO-OPERAZIONI

L'assemblatore ha le seguenti pseudo-operazioni fondamentali:

DEFB	–	**DEFINE BYTE**
DEFL	–	**DEFINE LABEL**
DEFM	–	**DEFINE STRING**
DEFS	–	**DEFINE STORAGE**
DEFW	–	**DEFINE WORD**
END	–	**END**
EQU	–	**EQUATE**
ORG	–	**ORIGIN**

PSEUDO-OPERAZIONI DEFB, DEFM, DEFW

DEFB, DEFM e DEFW sono pseudo-operazioni sul Dato usate per mettere dati nella ROM. DEFB è usato per dati ad 8 bit, DEFW per dati a 16 bit e DEFM per stringhe ASCII (lunghe 63 o meno carat-

teri). Il solo aspetto non usuale da ricordare è che DEFW immagazzina gli 8 bit meno significativi del dato nella prima parola e gli otto bit più significativi nella seconda parola. Questa è la procedura standard per 8080A/8089/Z80 per immagazzinare indirizzi in memoria, ma ciò va contro all'uso comune. Dovete stare attenti all'ordine con cui immagazzinate dati a 16 bit.

Si noti che DEFB e DEFW definiscono rispettivamente il valore di solo un singolo byte o di una singola parola. Lo stabilire una tabella di valori richiede una serie di pseudo-operazioni DEFB o DEFW, una per ogni byte o parola dati.

Esempi:

```
ADDR:   DEFW    3165H
```

ha come risultato (ADDR) = 65 e (ADDR + 1) = 31 (esadecimale)

```
TCONV:  DEFB    32
```

Questa pseudo-operazione posiziona il numero 32 nel successivo byte di ROM ed assegna il nome TCONV all'indirizzo di quel byte.

```
ERROR:  DEFM    'ERROR'
```

Questa pseudo-operazione posiziona i caratteri in ASCII a 7 bit E, R, R, O ed R nei successivi cinque byte di ROM ed assegna il nome ERROR all'indirizzo del primo byte.

```
OPERS:  DEFW    FADD
        DEFW    FSUB
        DEFW    FMUL
        DEFW    FDIV
```

Questa serie di pseudo-operazioni mette gli indirizzi FADD, FSUB, FMUL E FDIV nei successivi otto byte di memoria ed assegna il nome OPERS all'indirizzo del primo byte. Si noti che il primo byte contiene i bit meno significativi dell'indirizzo FADD.

PSEUDO-OPERAZIONE DEFS

DEFS è la pseudo-operazione Reserve usata per assegnare locazioni di RAM; essa alloca un numero specificato di byte.

PSEUDO-OPERAZIONE EQU

EQU è la pseudo-operazione Equate o Define usata per assegnare valori ai nomi.

PSEUDO-OPERAZIONE DEFL

DEFL è simile ad EQU, tranne che DEFL permette al nome di essere ridefinito dopo. DEFL è molto simile alla direttiva SET in altri assemblatori. Dovrebbe essere usata solo per definire variabili di tempo di assembly (cioè quelle variabili usate in assembly condizionato o in statement di espansione di macro condizionati).

PSEUDO-OPERAZIONE ORG

ORG è la pseudo-operazione standard Origin.

I programmi dello Z80 hanno di solito parecchie origini; le origini sono usate come segue:

1) Per specificare l'indirizzo di RESET (di solito zero).
2) Per specificare i punti d'ingressi delle interruzioni (di solito da 0 a 66_{16}, ma possono essere dovunque nella memoria).
3) Per specificare l'indirizzo di partenza del programma principale.
4) Per specificare gli indirizzi di partenza dei sottoprogrammi.

5) Per definire zone di immagazzinamento in RAM.
6) Per definire una zona di RAM per lo Stack.
7) Per specificare indirizzi usati per porte di I/O e per funzioni speciali.

Esempi:

```
RESET   EQU     0
        ORG     RESET
```

Questa sequenza posiziona la sequenza di istruzioni RESET in memoria a partire dall'indirizzo 0.

```
INT1    EQU     38H
        ORG     INT1
```

La sequenza di istruzioni che segue è memorizzata in memoria a partire dalla locazione 38_{16}.

PSEUDO-OPERAZIONE END

END indica semplicemente la fine del programma in linguaggio assembly. Le pseudo-operazioni a scopo speciale COND, MACRO, ENDIC e ENDM sono descritte più avanti in questo capitolo.

LABEL CON PSEUDO-OPERAZIONI

Le regole e le raccomandazioni per le label con pseudo-operazioni sono le seguenti:

1) EQU, DEFL e MACRO richiedono label, poiché la funzione di queste pseudo-operazioni è di definire il significato di quelle label.
2) DEFB, DEFM, DEFW e DEFS di solito hanno label.
3) ORG, COND, ENDC, ENDM ed END potrebbero non avere label, poiché il significato di tali label non è chiaro.

INDIRIZZI

NUMERI E CARATTERI NEL CAMPO INDIRIZZI

L'assemblatore Z80 della Zilog permette ingressi nel campo indirizzi in una delle forme seguenti:

1) Decimale (caso di default).
   Esempio: 1247
2) Esadecimale (deve cominciare con un digit e finisce con un H).
   Esempi: 142CH, 0E7H
3) Ottale (deve definire con O o Q, ma Q si confonde meno).
   Esempio: 1247Q o 1247O
4) Binario (deve definire con B).
   Esempio: 1001001000111B
5) ASCII (racchiuso tra apici).
   Esempio: 'HERE'
6) Come offset del Contatore di Programma ($).
   Esempio: $ + 237H

OPERAZIONI LOGICHE ED ARITMETICHE DI ASSEMBLATORE

Tutte le operazioni logiche ed aritmetiche in un campo indirizzi suppongono che tutti gli argomenti siano dati a 16 bit; questi producono risultati a 16 bit. Queste operazioni sono permesse come parti di espressioni nel campo indirizzi.

Quando si definiscono costanti di indirizzo, si dovrebbe usare la notazione esadecimale. Costanti binarie a 16 bit sono poco maneggevoli e quindi inclini ad errori. Costanti ottali non sono convenienti per il fatto che gli indirizzi sono memorizzati nel formato byte di ordine basso, byte di ordine alto. Questa divisione si verifica a metà di un digit ottale, cosa che provoca la spaccatura di un digit. Per esempio, per esprimere l'indirizzo 9D7FH o 116577Q nel formato basso-alto si ottiene 7F9DH o 77236Q. Come si può vedere, nella notazione esadecimale i digit sono semplicemente trasposti, mentre tale semplice rapporto non esiste per la notazione ottale.

OPERATORE	FUNZIONE	PRIORITÀ
+	PIÚ	1
-	MENO	1
.NOT. o\	NOT LOGICO	1
.RES.	RISULTATO	1
**	ELEVAZIONE A POTENZA	2
*	MOLTIPLICAZIONE	3
/	DIVISIONE	3
.MOD.	MODULO	3
.SHR.	SCORRIMENTO LOGICO A DESTRA	3
.SHL.	SCORRIMENTO LOGICO A SINISTRA	3
+	ADDIZIONE	4
-	SOTTRAZIONE	4
.AND o &	AND LOGICO	5
.OR. o ↑	OR LOGICO	6
.XOR.	XOR LOGICO	6
.EQ. o =	UGUALE	7
.GT. o >	MAGGIORE DI	7
.LT. o <	MINORE DI	7
.UGT.	VALORE SENZA SEGNO >	7
.ULT.	VALORE SENZA SEGNO <	7

Nelle espressioni di indirizzi con più di un operatore, l'ordine di valutazione è definito dalla priorità data nella lista precedente. Operatori aventi la stessa priorità sono valutati da sinistra a destra. Dapprima vengono valutate espressioni tra parentesi. Ricordate che una espressione chiusa interamente tra parentesi rappresenta un indirizzo di memoria.

Si noti ciò che segue:

1) L'operatore Result (.RES.) provoca l'eliminazione dell'overflow; cioè un cambiamento di segno provocato da un overflow nel bit di segno non risulta come errore dell'assemblatore.
2) Gli scorrimenti hanno la forma:

```
.SHR.    op1, op2
.SHL.    op1, op2
```

   dove op1 è il numero da far scorrere e op2 il numero degli scorrimenti. Gli scorrimenti sono logici; cioè si fanno scorrere degli Zero nei bit di peso alto o nei bit di peso basso, rispettivamente.
3) Gli operatori di confronto producono un risultato o di Vero logico (tutti uno) e di Falso logico (Zero).
4) Gli operatori .GT. ed .LT. presuppongono numeri di complemento a due con segno, mentre .UGT. ed .ULT. presuppongono operandi senza segno. Ciò significa che, per .GT. ed .LT., i numeri di complemento a due positivi sono maggiori di numeri di complemento a due negativi, mentre l'opposto si verifica per .UGT. ed .ULT.

ASSEMBLY CONDIZIONATO

PSEUDO-OPERAZIONI COND ED ENDC

L'assemblatore dello Z80 ha una semplice possibilità di assembly condizionato, basato sulle pseudo - operazioni COND e ENDC. COND è seguita da una espressione, per esempio:

```
COND    BASE - 1000H
        oppure
COND    BASE - OPER1
```

Se l'espressione non è zero, l'assemblatore include tutte le istruzioni fino alla pseudo-operazione ENDC del programma; se l'espressione è zero, l'assemblatore ignora tutte le istruzioni comprese tra COND e ENDC.

Non useremo assemblaggi condizionati o non ci riferiremo più a questa possibilità; talvolta è utile per aggiungere od eliminare istruzioni o configurare un'unica versione di un programma comune.

MACRO

PSEUDO-OPERAZIONI MACRO ED ENDM

L'assemblatore standard dello Z80 ha una possibilità di macro che assegna nomi a sequenze di istruzioni. Usate la pseudo-operazione MACRO per dare inizio alla definizione e ENDM per terminarla.
La macro può avere parametri e può includere qualunque istruzione in linguaggio assembly tranne le definizioni di altre macro.

La possibilità di macro è spesso una conveniente forma stenografica di programmazione, ma non la useremo.

Si noti che le sequenze di istruzioni definite da macro sono generalmente molto corte; non dovrebbero superare dieci o quindici istruzioni. Sequenze più lunghe dovrebbero essere fatte con sottoprogrammi per conservare spazio di memoria.

Ogni pseudo-operazione MACRO deve avere una label; la label è il nome con cui si identifica la macro. Per una trattazione di questo soggetto vedere il Capitolo 2.

Capitolo 4

PROGRAMMI SEMPLICI

L'unico modo per imparare il linguaggio di programmazione assembly è quello di scrivere programmi in questo linguaggio. Questo è ciò che faremo per i prossimi sei capitoli, i quali contengono esempi di tipici compiti del microprocessore. I problemi al termine di ciascun capitolo contengono delle varianti agli esempi forniti nel testo del capitolo. Dovreste cercare di rendere pratici gli esempi mediante un sistema a microcomputer basato sullo Z80, per assicurarvi di aver compreso il materiale descritto nel capitolo.

In questo capitolo cominciamo con alcuni programmi molto semplici:

FORMATO GENERALE DEGLI ESEMPI

FORMATO DEGLI ESEMPI

Ogni esempio di programma contiene le seguenti parti:

1) Un titolo che descrive il problema generale.
2) Una dichiarazione di scopo che descrive il compito specifico che il programma esegue, più le locazioni di memoria di cui esso fa uso.
3) Un problema campione che mostra i dati d'ingresso ed i risultati.
4) Un diagramma di flusso se la logica del programma è complessa.
5) Il programma sorgente oppure il listing in linguaggio assembly del programma.
6) Il programma oggetto oppure il listing in linguaggio di macchina esadecimale del programma.
7) Note esplicative che trattano le istruzioni ed i metodi usati nel programma.

I problemi al termine del capitolo sono simili agli esempi; i problemi dovrebbero essere programmati su un sistema a microcomputer basato sullo Z80 utilizzando gli esempi come guida.

I programmi sorgente negli esempi sono stati costruiti come segue:

DIRETTIVE PER GLI ESEMPI

1) Vengono usate le notazioni standard dell'assemblatore dello Zilog Z80, come riassunto nel Capitolo 3.
2) Le forme nelle quali dati e indirizzi appaiono sono scelte per la loro chiarezza piuttosto che per la coerenza. Si usano numeri esadecimali per gli indirizzi di memoria, i codici di istruzione e i dati BCD; numeri decimali per le costanti numeriche; numeri binari per le maschere logiche e numeri ASCII per i caratteri.
3) Sono poste in evidenza istruzioni usate di frequente e tecniche di programmazione.
4) Gli esempi illustrano i compiti che eseguono i microprocessori nelle comunicazioni, nelle strumentazioni, nei computer, nelle attrezzature aziendali, nelle applicazioni industriali e militari.
5) Sono inclusi commenti particolareggiati.
6) Sono evidenziate strutture semplici e chiare, ma i programmi sono i più efficienti possibili entro queste direttive. Le note spesso descrivono procedure più efficienti.
7) I programmi usano locazioni di memoria coerenti. Ogni programma parte dalla locazione di memoria 0000 (la locazione RESET) e termina con l'istruzione di HALT. Se il vostro microcomputer non ha monitor ed interrupt, potete scegliere di terminare i programmi con una istruzione ciclica infinita, ad esempio:

```
HERE: JR    HERE
```

La versione esadecimale è 18 seguito da FE. Potete sostituire l'istruzione HALT o JR HERE con l'istruzione RESTART o JP che trasferisce il controllo al monitor in alcuni microcomputer basati sullo Z80.

Consultate il manuale di utilizzc del vostro microcomputer per determinare le locazioni di memoria richieste e l'istruzione finale per il vostro sistema particolare.

DIRETTIVE PER I PROBLEMI

DIRETTIVE DI PROGRAMMAZIONE

Quando affrontate i problemi alla fine di ciascun capitolo, cercate di lavorare secondo le seguenti direttive:

1) Commentate ciascun programma in modo che altri possano comprenderlo. I commenti possono essere brevi e non grammaticali; debbono spiegare lo scopo di una sezione o istruzione nel programma. I commenti non debbono descrivere il funzionamento di istruzioni; questa descrizione è disponibile nei manuali. Non dovete commentare ogni affermazione o spiegare ciò che è ovvio. Potete seguire il formato degli esempi ma fornire meno dettagli.
2) Porrete in evidenza la chiarezza, la semplicità e la buona struttura nei programmi. Mentre questi ultimi debbono essere ragionevolmente efficienti, non preoccupatevi del risparmio del singolo byte di memoria di programma o di pochi microsecondi.
3) Rendete i programmi possibilmente generici. Non confondete i parametri (quali il numero di elementi in una matrice con costanti fisse) come π o C in ASCII).
4) Non assumete mai valori iniziali fissati per i parametri, vale a dire usate un'istruzione per caricare un valore iniziale dentro un parametro.
5) Usate notazioni assembler come mostrato negli esempi e definito nel Capitolo 3.
6) Usate notazioni esadecimali per gli indirizzi e la forma più chiara possibile per i dati.
7) Se il vostro microcomputer lo permette, comunicate tutti i programmi nella locazione di memoria 0000 e usate locazioni che partono da 0040_{16} per i dati ed immagazzinamenti temporanei. Altrimenti stabilite indirizzi equivalenti per il vostro microcomputer e usateli con coerenza. Inoltre, consultate il manuale per utilizzatore.
8) Adoperate nomi significativi per le label e le variabili, come SUM o CHECK invece che X, Y, o Z.
9) Eseguite ciascun programma sul vostro microcomputer. Non v'è altro modo di assicurarvi che il vostro programma sia corretto. Abbiamo fornito dati campione con ogni problema. Siate sicuri che il programma lavori per casi particolari.

Ora riassumiamo alcune utili informazioni che dovreste tenere a mente nella stesura dei programmi.

USO DELLO ACCUMULATORE

Quasi tutte le istruzioni di processo (es. ADD, SUBSTRACT, AND, OR) fanno uso dell'Accumulatore. Nella maggior parte dei casi caricherete il dato nell'accumulatore con LD, usando o LDA, (addr) per caricare il dato da una qualsiasi cella di memoria oppure LDA, (HL) per caricare il dato dall'indirizzo specificato nei Registri H e L. Ricordate che le parentesi indicano un indirizzo di memoria e non un dato.

USO DELLA COPPIA DI REGISTRI HL

Il metodo preferito per l'accesso alla memoria sta nell'uso di indirizzamenti implicati tramite i registri H e L, cioè adoperando (HL). Questo codice permette allo Z80 di eseguire un accesso alla memoria usando l'indirizzo contenuto nei Registri H ed L. Potete usare LD HL, data 16 per caricare un numero fisso nei registri H e L oppure LD HL, (addr) per caricare il contenuto di due celle successive di memoria in H e L. Potete usare INC HL oppure DEC HL per incrementare o decrementare (di 1) l'indirizzo nei Registri H e L.

Le operazioni logiche ed aritmetiche ad 8 bit utilizzano tutte il dato presente nell'Accumulatore come uno dei loro operandi e collocano i loro risultati nello stesso.

Alcune operazioni logiche ed aritmetiche ad 8 bit hanno usi particolari, per esempio:

SUB A (o XOR A) azzerano l'Accumulatore.

ADD A, A sposta verso sinistra di un bit l'Accumulatore in modo logico. Questa istruzione inoltre moltiplica il contenuto dell'Accumulatore per 2. AND A (o ORA) azzera il flag di Carry conservando il contenuto dell'Accumulatore.

Un AND logico può mascherare parti di una parola. La maschera richiesta ha i bit ad «1» nelle posizioni che volete conservare ed i bit a «0» nelle posizioni che volete azzerare.

ESEMPI DI PROGRAMMA

Complemento Ad Uno

Scopo: Complementare in modo logico il contenuto della cella di memoria 0040 e porre il risultato nella locazione 0041.

Problema Campione:

```
          (0040)   =6A
Risultato: (0041)  =95
```

Programma Sorgente:

```
LD      A,(40H)   ;PRENDI IL DATO
CPL               ;FAI IL COMPLEMENTO
LD      (41H),A   ;IMMAGAZZINA IL RISULTATO
HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)	
0000	3A	LD	A,(40H)
0001	40		
0002	00		
0003	2F	CPL	
0004	32	LD	(41H),A
0005	41		
0006	00		
0007	76	HALT	

Le istruzioni LDA, (addr), e LD (addr), A contengono indirizzi per determinare la provenienza o la destinazione del dato. Gli indirizzi sono costituiti da 16 bit, con gli otto bit meno significativi nella parola che segue immediatamente il codice d'istruzione e gli otto bit più significativi nella parola successiva (quest'ordine è contrario alla normale regola dei computer). CPL è un'istruzione di una parola che inverte ciascun bit dell'Accumulatore. Essa sostituisce tutti gli «0» con un «1» e gli «1» con uno «0», proprio come un blocco di porte invertenti.

Viene usata HALT al termine di tutti gli esempi.

Notate che potremmo anche porre un indirizzo nei Registri H e L e quindi usare quell'indirizzo durante tutto il programma.

Programma Sorgente:

```
LD      HL,40H   ;PUNTA SULL'OPERANDO
LD      A,(HL)   ;PRENDI IL DATO
CPL              ;FAI IL COMPLEMENTO
INC     HL       ;PUNTA SULLA DESTINAZIONE
LD      (HL),A   ;IMMAGAZZINA IL RISULTATO
HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)	
0000	21	LD	HL,40H
0001	40		
0002	00		
0003	7E	LD	A,(HL)
0004	2F	CPL	
0005	23	INC	HL
0006	77	LD	(HL),A
0007	76	HALT	

Quale versione pensate sia la migliore?

Le due versioni richiedono lo stesso numero di byte di memoria anche se la seconda è più lunga di due istruzioni. Ciò perchè la seconda versione usa meno indirizzi espliciti.

Somma Ad 8 Bit

Scopo: Sommare il contenuto delle celle di memoria 0040 e 0041, e porre il risultato nella locazione 0042.

Problema Campione:

```
            (0040)   =38
            (0041)   =2B
Risultato:  (0042)   =63
```

Programma Sorgente:

```
LD      A,(40H)  ;PRENDI IL PRIMO OPERANDO
LD      B,A      ;SALVA IL PRIMO OPERANDO
LD      A,(41H)  ;PRENDI IL SECONDO OPERANDO
ADD     A,B      ;SOMMA GLI OPERANDI
LD      (42H),A  ;IMMAGAZZINA LA SOMMA
HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)	
0000	3A	LD	A,(40H)
0001	40		
0002	00		
0003	47	LD	B,A
0004	3A	LD	A,(41H)
0005	41		
0006	40		
0007	80	ADD	A,B
0008	32	LD	(42H),A
0009	42		
000A	00		
000B	76	HALT	

Anche qui, potremmo usare alternativamente i registri H e L come sorgente di tutti gli indirizzi.

Programma Sorgente:

```
LD      HL,40H
LD      A,(HL)     ;PRENDI IL PRIMO OPERANDO
INC     HL
ADD     A,(HL)     ;SOMMA IL SECONDO OPERANDO
INC     HL
LD      (HL),A     ;IMMAGAZZINA IL RISULTATO
HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)	
0000	21	LD	HL,40H
0001	40		
0002	00		
0003	7E	LD	A,(HL)
0004	23	INC	HL
0005	86	ADD	A,(HL)
0006	23	INC	HL
0007	77	LD	(HL),A
0008	76	HALT	

In questo caso, il programma che fa uso dei registri H e L è più breve di quello che usa l'indirizzamento diretto. Perchè?

LD HL, 40H carica il contenuto delle due parole seguenti della memoria di programma nella coppia di Registri HL. La prima parola va nel Registro L, mentre la seconda nel registro H.

Il codice (HL) significa che il dato viene ottenuto dalla o inviato alla cella di memoria indirizzata dai Registri H e L. Pertanto, LD A, (HL) carica l'Accumulatore col contenuto della cella indirizzata; LD (HL), A carica la cella indirizzata col contenuto dell'Accumulatore. ADD A, (HL) somma il contenuto della cella indirizzata da HL al contenuto dell'Accumulatore. Ricordate che H e L contengono un indirizzo a 16 bit, ma la locazione di memoria che questi indirizzano contiene otto bit di dato. Notate la differenza fra ADD A, (HL) ADD A, H oppure ADD A, L.

INC HL esegue un incremento di 16 bit in un ciclo d'istruzione. La CPU non usa l'unità aritmetica ad 8 bit per l'incremento; essa utilizza l'incrementatore che usa normalmente per incrementare il Contatore di Programma a 16 bit.

LD A, (HL) e LD (HL), A sono preferibili a LD A, (addr) e LD (addr), A ogni volta che usate la stessa cella di memoria ripetutamente oppure celle adiacenti, dato che LD A, (HL) e LD (HL), A necessitano di minor tempo e memoria programmi. Notate, tuttavia, che dovete caricare i Registri H e L prima di poter usare (HL).

Spostamento di un Bit Verso Sinistra

Scopo: Spostare il contenuto della cella di memoria 0040 di un bit verso sinistra e porre il risultato nella cella di memoria 0041. Azzerare la posizione del bit evacuato. Questo tipo di spostamento è noto come «shift» logico. In uno «shift» logico, viene sempre forzato il valore zero.

Problema Campione:

```
          (0040)   =6F
Risultato: (0041)  =DE
```

Programma Sorgente:

```
LD      A,(40H)   ;PRENDI IL DATO
ADD     A,A       ;SCORRIMENTO A SINISTRA
LD      (41H),A   ;IMMAGAZZINA IL RISULTATO
HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)	
0000	3A	LD	A,(40H)
0001	40		
0002	00		
0003	87	ADD	A,A
0004	32	LD	(41H),A
0005	41		
0006	00		
0007	76	HALT	

ADD A, A somma semplicemente il contenuto dell'Accumulatore a se stesso. Il risultato, naturalmente, è il doppio del dato originale, questo è lo stesso risultato che produrrebbe uno «shift» logico. Il bit meno significativo del risultato è zero, dal momento che 0 + 0 = 1 + 1 = 0; 1 + 1 produce inoltre un riporto (Carry) sul successivo bit.

Alternativamente, noi possiamo sostituire ADD A, A con SLA A, che certamente è la scelta più ovvia. Tuttavia, SLA A richiede due parole di memoria di programma ed otto cicli di clock, mentre ADD A, A richiede una parola e quattro cicli di clock. La differenza è causata dal fatto che SLA A è un'istruzione supplementare aggiunta alla serie originale dell'8080A (ricordate il confronto presentato precedentemente).

Mascheratura Del Quattro Bit Più Significativi

Scopo: Porre i quattro bit meno significativi della cella di memoria 0040 nei quattro bit meno significativi della cella 0041. Azzerare i quattro bit più significativi della cella di memoria 0041.

Problema Campione:

(0040) =3D
Risultato (0041) =0D

Programma Sorgente:

```
LD      A,(40H)       ;PRENDI IL DATO
AND     00001111B     ;MASCHERA I 4 BIT MENO SIGNIFICATIVI
LD      (41H),A       ;IMMAGAZZINA IL RISULTATO
HALT
```

Nota: B significa binario nella notazione standard dell'assemblatore Z80.

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)	
0000	3A	LD	A,(40H)
0001	40		
0002	00		
0003	E6	AND	00001111B
0004	0F		
0005	32	LD	(41H),A
0006	41		
0007	00		
0008	76	HALT	

La maschera (00001111) viene scritta in binario per rendere la sua funzione più chiara al lettore. La notazione binaria per le maschere è generalmente più comprensibile di quella esadecimale, benché i risultati siano gli stessi. La notazione esadecimale deve essere utilizzata per maschere più lunghe di quattro bit. I commenti debbono spiegare l'operazione di mascheratura.

Quando l'argomento nel campo indirizzo è un numero, AND esegue la funzione logica AND del contenuto dell'Accumulatore con il contenuto della parola di memoria di programma immediatamente successiva all'istruzione. AND può essere usato per azzerare bit che non si usano. I quattro bit meno significativi potrebbero essere un ingresso proveniente da un interruttore o un'uscita verso un visualizzatore numerico.

Azzeramento Di Una Locazione Di Memoria

Scopo: Azzerare la locazione di memoria 0040.

Programma Sorgente:

```
SUB     A
LD      (40H),A   ;AZZERA LA LOCAZIONE 40
HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)
0000	97	SUB A
0001	32	LD (40H),A
0002	40	
0003	00	
0004	76	HALT

SUB A sottrae il numero nell'Accumulatore a se stesso. Il risultato è l'azzeramento dell'Accumulatore. SUB A, XOR A, oppure LD A,0 possono azzerare l'accumulatore. LD A,0 necessita di più tempo e memoria ma non modifica i «flag» di stato.

Frazionamento Della Parola

Scopo: Dividere il contenuto della cella di memoria 0040 in due gruppi di 4 bit e immagazzinarli nelle celle di memoria 0041 e 0042. Porre i quattro bit più significativi della cella di memoria 0040 nelle posizioni dei quattro bit meno significativi della cella 0041; porre i quattro bit meno significativi della cella 0040 nelle posizioni dei quattro bit meno significativi della cella di memoria 0042. Azzerare le posizioni dei 4 bit più significativi delle celle di memoria 0041 e 0042.

Problema Campione:

```
          (0040)   =3F
Risultato: (0041)  =03
          (0042)   =0F
```

Programma Sorgente:

```
LD      HL,40H
LD      A,(HL)        ;PRENDI IL DATO
LD      B,A
RRA                   ;FAI SCORRERE IL DATO A SINISTRA 4 VOLTE
RRA
RRA
RRA
AND     00001111B     ;MASCHERA I BIT PIÙ SIGNIFICATIVI
INC     HL
LD      (HL),A        ;IMMAGAZZINA I BIT PIÙ SIGNIFICATIVI
LD      A,B           ;RIIMMAGAZZINA I DATI ORIGINALI
AND     00001111B     ;MASCHERA I BIT MENO SIGNIFICATIVI
INC     HL
LD      (HL),A        ;IMMAGAZZINA I BIT MENO SIGNIFICATIVI
HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)	
0000	21	LD	HL,40H
0001	40		
0002	00		
0003	7E	LD	A,(HL)
0004	47	LD	B,A
0005	1F	RRA	
0006	1F	RRA	
0007	1F	RRA	
0008	1F	RRA	
0009	E6	AND	00001111B
000A	0F		
000B	23	INC	HL
000C	77	LD	(HL),A
000D	78	LD	A,B
000E	E6	AND	00001111B
000F	0F		
0010	23	INC	HL
0011	77	LD	(HL),A
0012	76	HALT	

Le istruzioni che usano gli indirizzi nei Registri H e L occupano solo una parola di memoria programmi. Tuttavia, HL devono essere caricati prima di poter usare l'indirizzo. Così, l'indirizzamento di memoria implicito fa risparmiare tempo e memoria, se paragonato all'indirizzamento di memoria diretto, solo quando il programma fa uso più volte dello stesso indirizzo o d'indirizzi consecutivi.

RRA sposta l'Accumulatore verso destra di un bit in modo circolare, ponendo il bit meno significativo al posto di quello più significativo e nel Carry. Lo spostamento verso destra dell'Accumulatore per quattro volte necessita di quattro istruzioni RRC. Potremmo usare SRL A per procurare direttamente uno «shift» logico (non sarebbe allora necessario alcun AND finale). Comunque SRL A richiede una doppia quantità di memoria e di tempo rispetto a RRC. Provate a sostituire SRL A con RRC e notate la differenza. Un'altra alternativa è quella di usare l'istruzione RLD per sostituire sia la maschera che la memorizzazione. Tuttavia, questa soluzione non è ottimale sia dal punto di vista della memorizzazione che della velocità di esecuzione a causa del vincolo che il semibyte (nibble) di ordine elevato di ciascun risultato deve essere uguale a zero.

Molte istruzioni dello Z80 agiscono su coppie di registri ad 8 bit. Le coppie sono HL (H e L), DE (D e E), e BC (B e C). I registri B, D e H sono gli otto bit più significativi della coppia; i Registri C, E e L sono gli otto bit meno significativi. Le comuni istruzioni che usano coppie di registri sono LD rp (Carica la Coppia di Registri), INC rp (Incrementa la Coppia di Registri), DEC rp (Decrementa la Coppia di Registri) e ADD HL, rp (Somma la Coppia di Registri ad H e L).

Ricerca del Maggiore di Due Numeri

Scopo: Porre il maggiore dei contenuti delle celle di memoria 0040 e 0041 nella cella 0042. Supponiamo che il contenuto delle locazioni di memoria 0040 e 0041 siano numeri binari senza segno.

Problemi Campione:

```
a.            (0040)   =3F
              (0041)   =2B
   Risultato: (0042)   =3F
b.            (0040)   =75
              (0041)   =A8
   Risultato: (0042)   =A8
```

Programma Sorgente:

```
        LD      HL,40H
        LD      A,(HL)    ;PRENDI IL PRIMO OPERANDO
        INC     HL
        CP      (HL)      ;IL SECONDO OPERANDO È MAGGIORE?
        JR      C,DONE
        LD      A,(HL)    ;SÌ, PRENDI IL SECONDO OPERANDO
DONE:   INC     HL
        LD      (HL),A    ;IMMAGAZZINA L'OPERANDO MAGGIORE
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)		
0000	21		LD	HL,40H
0001	40			
0002	00			
0003	7E		LD	A,(HL)
0004	23		INC	HL
0005	BE		CP	(HL)
0006	30		JR	NC,DONE
0007	01			
0008	7E		LD	A,(HL)
0009	23	DONE:	INC	HL
000A	77		LD	(HL),A
000B	76		HALT	

CP (HL) posiziona i «flag» come se il contenuto della cella di memoria indirizzata da H e L fosse stata sottratta dal contenuto dell'Accumulatore. Tuttavia quest'ultimo resta invariato per le successive comparazioni o per altre procedure.

Se A è il contenuto dell'Accumulatore e X è il secondo operando per un'istruzione CP, allora i flag sono disposti come segue:

1) Zero = 1 se A = X
   Zero = 0 se A # X
2) Carry = 1 se A < X
   Carry = 0 se A ≥ X
   (A, X sono numeri binari privi di segno)

CP porta il Carry ad 1 se si rende necessario un prestito (borrow) per eseguire realmente la sottrazione, cioè se il numero da sottrarre al contenuto dell'Accumulatore è maggiore di quest'ultimo. Così la sequenza CP, JR NC, DONE porta ad un salto a DONE se il contenuto dell'Accumulatore è maggiore o uguale all'altro numero.

JR NC, DONE provoca un salto alla locazione di memoria DONE se il flag Carry = 0. Altrimenti (se Carry = 1), il computer continua con la locazione di memoria sequenzialmente adiacente alla istruzione JR.

DONE è una label, un nome che voi assegnate ad una cella in memoria in modo che sia più facile rammentarla. Notate che le label sono seguite dai due punti sulla riga dove vengono definite.

La label rende la destinazione del salto più chiara, in modo particolare quando si stanno per usare indirizzamenti relativi. L'assemblatore calcola lo spostamento richiesto (attenzione: alcuni assemblatori Z80 non eseguono questa operazione). Se si usa una label è preferibile specificare esattamente la destinazione (cioè JR NC, $ + 3) dato che le istruzioni dello Z80 sono di varia lunghezza. Potreste quindi facilmente commettere un errore nel determinare un salto.

Se le condizioni di salto non sono soddisfatte, il processore prosegue semplicemente verso la cella sequenzialmente successiva nella memoria programmi (cioè esegue l'istruzione LD A, (HL)).

L'assemblatore dello Z80 permette sei caratteri nelle label – il primo deve essere una lettera, mentre gli altri possono essere lettere o numeri (sono ammessi alcuni caratteri speciali ma noi non li useremo).

L'istruzione JR usa l'indirizzamento relativo nel quale la seconda parola dell'istruzione è il complemento a due di un numero ad 8 bit che la CPU somma all'indirizzo dell'istruzione seguente per trovare l'indirizzo di destinazione. Nell'esempio, lo spostamento relativo è 0009 (indirizzo di destinazione) meno 0008 (indirizzo immediatamente successivo al salto) vale a dire 01.

Si deve notare che alcuni assemblatori dello Z80 non calcoleranno lo spostamento nella forma mostrata. Questi assemblatori richiedono lo spostamento nel campo indirizzo, piuttosto che la label dell'istruzione di destinazione. Se possedete un tale assemblatore, usate la forma JR NC, DONE – $. Ricordate che $ significa «l'indirizzo dell'istruzione in corso».

Lo Z80 ha due gruppi d'istruzione di salto, JP (Jump) e JR (Jump Relativo). Le istruzioni JP richiedono un indirizzo completo di memoria; esse occupano tre byte di memoria e si eseguono in dieci cicli di clock. Le istruzioni JR richiedono solo uno spostamento di una parola; esse occupano due bit di memoria e si eseguono in 12 cicli se un salto viene effettivamente compiuto e in 7 cicli se non viene eseguito. Così le istruzioni JR utilizzano meno memoria delle istruzioni JP ma possono richiedere un po' di tempo in più se si compie un salto (il tempo supplementare viene usato per eseguire la somma richiesta a 16 bit del contatore di programma e dello spostamento).

Addizione a 16 Bit

Scopo: Sommare il numero a 16 bit nelle celle di memoria 0040 e 0041 al numero nelle celle 0042 e 0043. Gli otto bit più significativi sono nelle celle 0041 e 0043. Memorizzare il risultato nelle locazioni di memoria 0044 e 0045, con i bit più significativi in 0045.

Problema Campione:

```
          (0040) = 2A
          (0041) = 67
          (0042) = F8
          (0043) = 14
Risultato: 672A + 14F8 = 7C22
          (0044) = 22
          (0045) = 7C
```

Programma Sorgente:

```
LD      HL,(40H) ;PRENDI IL PRIMO NUMERO A 16 BIT
LD      DE,(42H) ;PRENDI IL SECONDO NUMERO A 16 BIT
ADD     HL,DE    ;ADDIZIONA A 16 BIT
LD      (44H),HL ;IMMAGAZZINA IL RISULTATO A 16 BIT
HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)	
0000	2A	LD	HL,(40H)
0001	40		
0002	00		
0003	ED	LD	DE,(42H)
0004	5B		
0005	42		
0006	00		
0007	19	ADD	HL,DE
0008	22	LD	(44H),HL
0009	44		
000A	00		
000B	76	HALT	

LD HL, (addr) carica i registri H e L da due locazioni di memoria, quella specificata nell'istruzione e quella consecutiva. Il contenuto della cella successiva va nel Registro H. Così, LD HL, (40H) significa L = (40), H = (41). Il reale trasferimento si effettua con un byte alla volta e necessita di 16 cicli di clock. Il vantaggio dell'istruzione di Caricamento a 16 bit rispetto a quelle ad 8 bit è che la CPU deve prelevare solo una istruzione dalla memoria.

Notate la differenza fra LD HL, (addr), che carica il contenuto delle due celle di RAM corrispondenti a addr e addr + 1 in H e L, LD HL, data 16, che carica il contenuto dei due byte seguenti indicati dal contatore d'istruzioni in H e L. Dato che questi due byte seguono immediatamente il codice operativo, caricamenti di questo tipo vengono definiti come istruzioni di caricamento immediato.

LD DE, (addr) è simile a LD HL, (addr) con l'eccezione che ci vuole una parola in più di memo-

ria e quattro cicli in più di clock. Questa è un'istruzione che è presente nel set dello Z80 ma non lo è in quello dell'8080/8085. Una soluzione alternativa è:

```
EX      DE, HL    ;CONSERVA IL PRIMO NUMERO A 16 BIT IN DE
LD      HL, (42H) ;PRENDI IL SECONDO NUMERO A 16 BIT
```

EX DE, HL scambia il contenuto dei Registri D e E con H e L. Nessun numero viene variato o cancellato. Il vantaggio di EX DE, HL diventerà evidente se provate a sostituirlo con una serie di istruzioni LD.

ADD HL, DE somma il numero a 16 bit nei Registri D e E al numero a 16 bit nei Registri H e L. Il risultato è posto nei Registri H e L. ADD HL, DE in verità somma un byte per volta. Essa viene eseguita in 11 cicli di clock.

LD (addr), HL immagazzina il contenuto dei Registri H e L in due celle di memoria, quella specificata nell'istruzione e quella consecutiva a quest'ultima. Il contenuto di L va nella locazione specificata ed il contenuto di H nella cella successiva. Così, LD (44H), HL significa (44) = L, (45) = H. Come con LD HL (addr) il trasferimento avviene realmente un byte alla volta e richiede 16 cicli di clock.

Sebbene lo Z80 sia un processore ad 8 bit, esso possiede delle istruzioni che trattano numeri di 16 bit. Queste istruzioni sono destinate principalmente per trattare indirizzi, ma potete anche usarle per dati a 16 bit. Le più comuni con il loro uso sono:

1) ADD HL, rp – Somma a 16 Bit
   Usata per accedere a tabella e per sommare unità di dati a 16 bit.
2) DEC rp – Decremento a 16 Bit
   Usata per sottrarre uno dal contenuto di una coppia di registri.
3) INC rp – Incremento a 16 Bit
   Usata per sommare uno al contenuto di una coppia di registri.
4) LD rp, data 16 – Caricamento immediato a 16 Bit
   Usato per inizializzare una coppia di registri con un valore fissato, ad es. l'indirizzo di partenza di una matrice o tabella.
5) LD HL, (addr) – Caricamento Diretto a 16 Bit di HL
   Usato per collocare indirizzi diversi nel registro indirizzi principale (H e L).
6) LD (addr), HL – Memorizzazione Diretta a 16 Bit di HL
   Usato per immagazzinare indirizzi in memoria dal registro indirizzi principale (H e L).

Tabella di Quadrati

Scopo: Calcolare il quadrato del contenuto della cella di memoria 0041 da una tabella e porre il risultato nella cella 0042. Supponiamo che la cella 0041 contenga un numero compreso tra 0 e 7 ($0 \leq (0041) \leq 7$).

La tabella occupa le locazioni di memoria da 0050 a 0057.

Indirizzo di Memoria (Esadecimale)	Ingresso	
	(Esadecimale)	(Decimale)
0050	00	0 (0^2)
0051	01	1 (1^2)
0052	04	4 (2^2)
0053	09	9 (3^2)
0054	10	16 (4^2)
0055	19	25 (5^2)
0056	24	36 (6^2)
0057	31	49 (7^2)

Problemi Campione:

a. (0041) =03
Risultato: (0042) =09
b. (0041) =06
Risultato: (0042) =24

Programma Sorgente:

```
        LD      A,(40H)     ;PRENDI IL DATO
        LD      L,A         ;ATTRIBUISCI IL DATO AD UN INDICE
                            ;  A 16 BIT
        LD      H,0
        LD      DE,SQTAB    ;PRENDI L'INDIRIZZO DI PARTENZA
                            ;  DELLA TABELLA
        ADD     HL,DE       ;TABELLA DEGLI INDICI CON DATI
        LD      A,(HL)      ;PRENDI IL QUADRATO DEL DATO
        LD      (41H),A
        HALT

        ORG     50H         ;TABELLA DEI QUADRATI
SQTAB:  DEFB    0
        DEFB    1
        DEFB    4
        DEFB    9
        DEFB    17
        DEFB    25
        DEFB    36
        DEFB    49
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)		
0000	3A		LD	A,(40H)
0001	40			
0002	00			
0003	6F		LD	L,A
0004	26		LD	H,0
0005	00			
0006	11		LD	DE,SQTAB
0007	50			
0008	00			
0009	19		ADD	HL,DE
000A	7E		LD	A,(HL)
000B	32		LD	(41H),A
000C	41			
000D	00			
000E	76			
0050	00	SQTAB:	DEFB	0
0051	01		DEFB	1
0052	04		DEFB	4
0053	09		DEFB	9
0054	10		DEFB	16
0055	19		DEFB	25
0056	24		DEFB	36
0057	31		DEFB	49

Notate che dovete anche far entrare in memoria la tabella dei quadrati (la pseudo-funzione DEFB dell'accumulatore tratterà ciò). La tabella dei quadrati è costituita da dati costanti e non da parametri che possono variare; ciò perchè possiate inizializzare la tabella usando la pseudo-funzione DEFB, invece che eseguendo istruzioni che carichino dei valori nella tabella. Ricordate che la tabella fa parte della memoria programmi (ROM nella maggior parte dei sistemi).

LD L, A sposta il dato dell'Accumulatore nel Registro L. Il dato è costituito dagli otto bit meno significativi dell'indice. Non potete sempre presumere che il dato presentato al vostro programma sia nel giusto campo di esecuzione. È sempre una buona regola controllare il campo di variazione di tutti i valori critici. Il controllo del campo di variazione consiste nel sondare un valore per assicurare che esso sia entro i giusti limiti superiore e inferiore. Qualsiasi byte può avere un valore nel campo da 0 a 255. Se il valore memorizzato nel byte alla locazione 0040 H è maggiore di sette, il programma farà riferimento a un byte indefinito oltre il termine della tabella dei quadrati, portando alla generazione nel programma di risultati sbagliati. Il controllo del campo di variazione eviterà il presentarsi di questo errore.

LD H, 0 azzera il Registro H cosicché non interferisca con la somma a 16 bit dell'indirizzo di partenza con l'indice. Non supporrete mai che un registro contenga Zero all'inizio di un programma.

LD DE, SQTAB carica l'indirizzo di partenza della tabella nei Registri D ed E. Si usano D ed E come indirizzo di partenza dato che l'istruzione ADD HL non fa variare D ed E. Così, l'indirizzo di partenza della tabella sarà ancora in D ed E dopo la somma, nel caso si voglia un altro elemento della tabella.

ADD HL, DE somma l'indirizzo di partenza e l'indice; il risultato in H e L è così l'indirizzo dal dato corretto. LD A, (HL) quindi porta quel dato nell'Accumulatore.

Le operazioni aritmetiche che un microprocessore non può eseguire direttamente con alcune istruzioni vengono spesso meglio effettuate mediante tabelle di riferimento. Le tabelle di riferimento contengono semplicemente tutte le possibili risposte al problema; sono organizzate in modo che la risposta ad un particolare problema possa essere facilmente individuata. Il problema aritmetico diventa ora un problema di accesso — come otterremo la corretta risposta dalla tabella? Dobbiamo conoscere due cose: la posizione della risposta nella tabella (chiamata indice) e l'indirizzo di base, o partenza della tabella. L'indirizzo dalla risposta è pari quindi all'indirizzo di base più l'indice.

L'indirizzo di base è naturalmente un numero fisso per una particolare tabella. Come possiamo determinare l'indice? Nei casi semplici, dove viene interessato un solo gruppo di dati, noi possiamo organizzare la tabella in modo tale che il dato sia anche l'indice. Nella tabella dei quadrati, l'ingresso Oesimo nella tabella contiene 0 al quadrato, il primo ingresso uno al quadrato, ecc. Nei casi più complessi, dove la diffusione dei valori d'ingresso è molto grande e dove sono coinvolti diversi elenchi di dati (ad es., radici quadrate o numero di permutazioni), dobbiamo usare metodi più complicati per determinare gli indici.

L'aspetto fondamentale nell'uso di una tabella è il tempo rispetto alla memoria. Le tabelle sono più veloci, poiché non sono richiesti calcoli, e sono più semplici, dato che non deve essere escogitato e collaudato alcun metodo matematico. Tuttavia, le tabelle possono occupare una grande quantità di memoria se il campo d'esecuzione dei dati di ingresso è ampio. Possiamo spesso ridurre le dimensioni di una tabella limitando la precisione dei risultati, riducendo opportunamente i dati d'ingresso, od organizzando intelligentemente la tabella. Le tabelle vengono spesso usate per calcolare funzioni trascendenti e trigonometriche, ingressi linearizzati, codici di conversione, e per eseguire altre funzioni matematiche.

Complemento ad Uno di Numeri a 16 Bit

Scopo: Posizionare il complemento ad uno del numero a 16 bit contenuto nelle celle di memoria 0040 e 0041 nelle locazioni di memoria 0042 e 0043. I byte più significativi sono nelle celle 0041 e 0043.

Problema Campione:

```
          (0040)   =67
          (0041)   =E2
Risultato: (0042)  =98
          (0043)   =1D
```

Il complemento ad uno inverte ciascun bit del numero originale; la somma del numero originale col suo complemento darà sempre tutti i bit uguali a 1.

Programma Sorgente:

```
LD      HL,(40H) ;PRENDI IL DATO
LD      A,L      ;FAI IL COMPLEMENTO DEGLI 8 BIT
                 ;   MENO SIGNIFICATIVI
CPL
LD      L,A
LD      A,H      ;FAI IL COMPLEMENTO DEGLI 8 BIT
                 ;   PIÙ SIGNIFICATIVI
CPL
LD      H,A
LD      (40H),HL ;IMMAGAZZINA I COMPLEMENTI AD UNO
HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)	
0000	2A	LD	HL,(40H)
0001	40		
0002	00		
0003	7D	LD	A,L
0004	2F	CPL	
0005	6F	LD	L,A
0006	7C	LD	A,H
0007	2F	CPL	
0008	67	LD	H,A
0009	22	LD	(42H),HL
000A	42		
000B	00		
000C	76	HALT	

Malgrado le istruzioni a 16 bit dello Z80, voi potete usare istruzioni ad 8 bit per eseguire la maggior parte delle operazioni aritmetiche e logiche. Le istruzioni a 16 bit possono, tuttavia, essere usate per caricare e memorizzare dati e occasionalmente per eseguire alcune operazioni aritmetiche a 16 bit, quali la somma, la differenza, l'incremento e il decremento. Imparerete presto che le istruzioni a 16 bit sono ben lungi dall'essere una serie completa e potete spesso incorrere in problemi imbarazzanti se le usate per maneggiare dati a 16 bit.

PROBLEMI

1) Complemento a Due:

Scopo: Porre il complemento a due del contenuto della cella di memoria 0040 nella cella 0041. Il complmento a due è il complemento a uno più uno.

Problema Campione:

(0040) = 3E
Risultato: (0041) = C2

La somma del numero originale e del suo complemento a due è 0 (provate il problema campione).

2) Sottrazione ad 8 Bit

Scopo: Sottrarre il contenuto della cella di memoria 0041 dal contenuto della cella 0040. Porre il risultato nella cella 0042.

Problema Campione:

(0040) = 77
(0041) = 39
Risultato: (0042) = 3E

3) Spostamento Verso Sinistra di Due Bit

Scopo: Spostare il contenuto della cella di memoria 0040 verso sinistra di due bit e porre il risultato nella cella 0041. Azzerare le posizioni dei due bit meno significativi.

Problema Campione:

(0040) = 5D
Risultato: (0041) = 74

4) Mascheratura dei Quattro Bit Meno Significativi

Scopo: Porre i quattro bit più significativi del contenuto della cella di memoria 0040 nella cella 0041. Azzerare i quattro bit meno significativi della locazione di memoria 0041.

Problema Campione:

(0040) = C4
Risultato: (0041) = C0

5) Posizionamento ad Uno di Tutti I Bit di una Locazione di Memoria

Scopo: I bit della locazione di memoria 0040 vengono portati tutti ad uno (FF esadecimale).

6) Composizione di una Parola

Scopo: Unire i quattro bit meno significativi delle celle di memoria 0040 e 0041 in una parola e memorizzarli nella cella 0042. Porre i quattro bit meno significativi della locazione di memoria 0040 nelle posizioni dei quattro bit più significativi della cella 0042; porre i quattro bit meno significativi della locazione 0041 nelle posizioni dei quattro bit meno significativi della cella 0042.

Problema Campione:

(0040) = 6A
(0041) = B3
Risultato: (0042) = A3

7) Ricerca del Minore di Due Numeri

Scopo: Porre il minore dei contenuti delle locazioni 0040 e 0041 nella cella di memoria 0042. Supporre che 0040 e 0041 contengano numeri binari privi di segno.

Problemi Campione:

a. (0040) = 3F
(0041) = 2B
Risultato: (0042) = 2B
b. (0040) = 75
(0041) = A8
Risultato: (0042) = 75

8) Addizione a 24 Bit

Scopo: Sommare il numero a 24 bit contenuto nelle celle 0040, 0041 e 0042 al numero a 24 bit nelle celle 0043, 0044 e 0045. Gli otto bit più significativi sono nelle celle 0042 e 0045; gli otto bit meno significativi sono nelle celle di memoria 0040 e 0043. Memorizzare il risultato nelle celle 0046, 0047 e 0048 con i bit più significativi in 0048 e i meno significativi in 0046.

Problema Campione:

```
            (0040) = 2A
            (0041) = 67
            (0042) = 35
            (0043) = F8
            (0044) = A4
            (0045) = 51
Risultato:  (0046) = 22
            (0047) = 0C
            (0048) = 87
cioè      35672A
        + 51A4F8
          870C22
```

9) Somma di Quadrati

Scopo: Calcolare i quadrati dei contenuti delle celle di memoria 0040 e 0041 e sommarli insieme. Porre il risultato nella cella di memoria 0042. Supponiamo che le celle di memoria 0040 e 0041 contengano ambedue numeri compresi fra 0 e 7 ($0 \leq (0040) \leq 7$ e $0 \leq (0041) \leq 7$). Usare le tabelle dei quadrati dell'esempio intitolato Tabella di Quadrati.

Problema Campione:

```
            (0040) = 03
            (0041) = 06
Risultato:  (0042) = 2D
cioè        3² + 6² = 9 + 36 = 45 (decimale)
                    = 2D (esadecimale)
```

10) Complemento a Due di una Parola a 16 Bit

Scopo: Porre il complemento a due del numero a 16 bit contenuto nelle locazioni di memoria 0040 e 0041 (i bit più significativi in 0041) nelle celle 0042 e 0043 (i bit più significativi in 0043).

Problemi Campione:

```
a.          (0040) = 00
            (0041) = 58
Risultato:  (0042) = 00
            (0043) = A8
b.          (0040) = 72
            (0041) = 00
Risultato:  (0042) = 8E
            (0043) = FF
```

Capitolo 5

SEMPLICI CICLI DI PROGRAMMA

Il ciclo di programma è quella struttura fondamentale che spinge la CPU a ripetere una sequenza di istruzioni. I cicli posseggono quattro parti:

1) La parte di inizializzazione, che stabilisce i valori di partenza dei contatori, dei registri indirizzi (puntatori) e delle altre variabili.
2) La parte di elaborazione, dove avviene la reale manipolazione dei dati. Questa è la parte che esegue l'opera.
3) La parte di controllo di ciclo, che aggiorna i contatori e i puntatori per la successiva ripetizione.
4) La parte conclusiva, che analizza e memorizza i risultati.

Notate che il computer esegue le Parti 1 e 4 una volta sola, mentre può eseguire le Parti 2 e 3 parecchie volte. Così, il tempo di esecuzione del ciclo dipenderà principalmente dal tempo di esecuzione delle Parti 2 e 3. Voi vorrete che le Parti 2 e 3 vengano eseguite il più velocemente possibile; non preoccupatevi del tempo di esecuzione delle Parti 1 e 4. Un tipico ciclo di programma può essere rappresentato in un diagramma di flusso come mostrato in Fig. 5-1, oppure le posizioni del processo e le parti di controllo di ciclo possono essere invertite come mostrato in Fig. 5-2. La parte di processo in Fig. 5-1 viene sempre eseguita almeno una volta, mentre la parte di processo in Fig. 5-2 non può essere eseguita affatto. La Fig. 5-1 sembra più naturale, ma la Fig. 5-2 è spesso più efficace ed evita il problema di cosa fare quando non ci sono dati (uno «spauracchio» per i computer è la causa frequente di insignificanti situazioni, come il computer che sollecita qualcuno con insistenza per il pagamento di $ 0,00).

La struttura a ciclo può essere usata per eseguire il processo su blocchi completi di dati. Per realizzare questo, il programma deve incrementare un registro indirizzi (solitamente la coppia di registri HL) dopo ogni ripetizione cosicché il registro indirizzi punti l'elemento successivo nel blocco dei dati. La successiva ripetizione eseguirà quindi le stesse operazioni sul dato nella cella di memoria seguente. Il computer può trattare blocchi di qualsiasi lunghezza con la stessa serie di istruzioni.

L'indirizzamento implicito tramite coppie di registri (in particolare HL) è la chiave per eseguire il processo su un blocco di dati con lo Z80, poiché ciò vi permette di variare l'effettivo indirizzo di memoria cambiando il contenuto dei registri. L'indirizzamento indicizzato, se è più lungo e più lento sullo Z80 dell'indirizzamento implicito, può essere utile quando si esegue il processo su più di un blocco di dati. Notate che nei modi di indirizzamento immediato e diretto gli indirizzi che vengono usati sono completamente determinati dall'istruzione (e così fissati se la memoria programmi è di sola lettura).

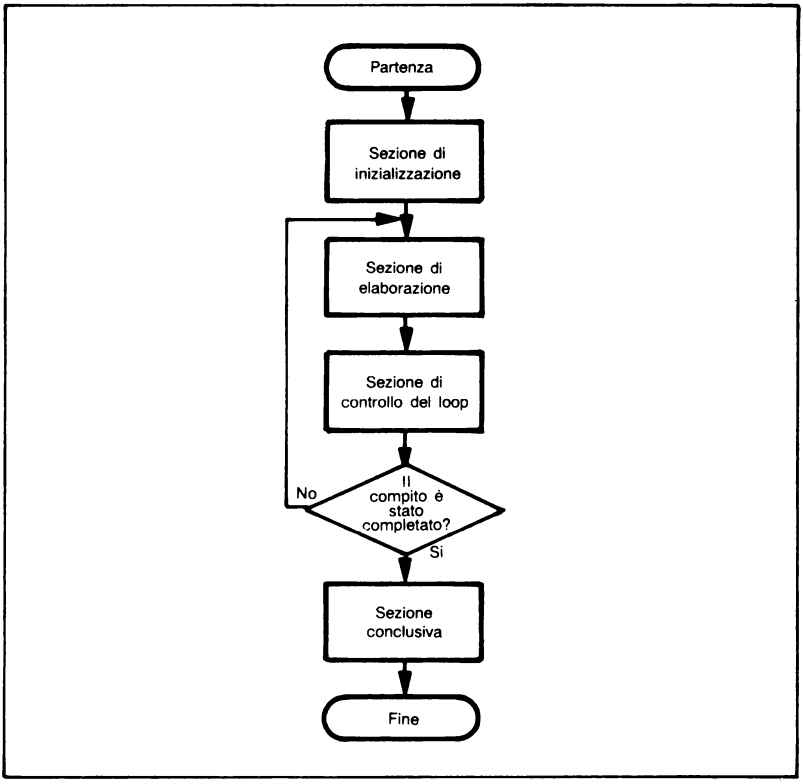


Figura 5-1. Diagramma di flusso di un Loop di Programma

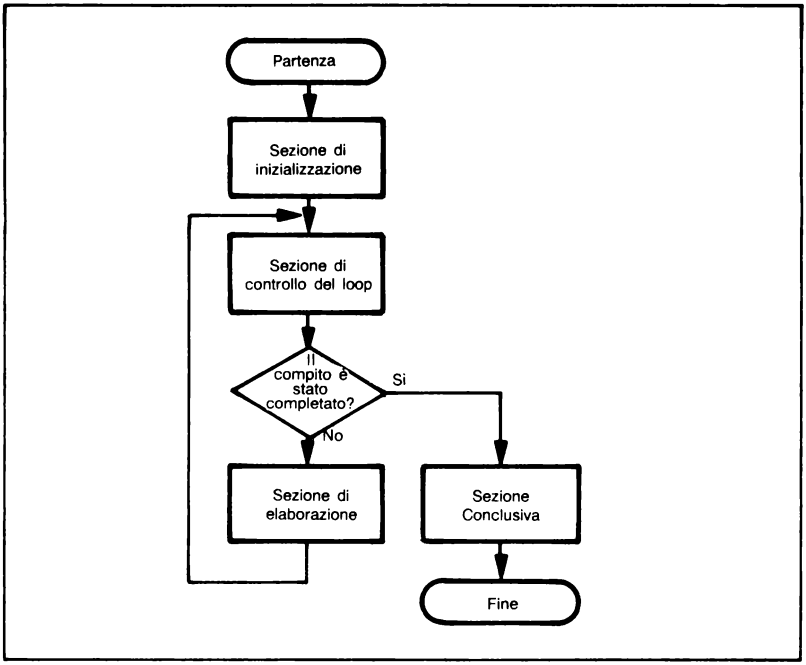


Figura 5-2. Un loop di Programma che permette Zero Iterazioni

ESEMPI

Somma di Dati

SOMMA AD 8 BIT

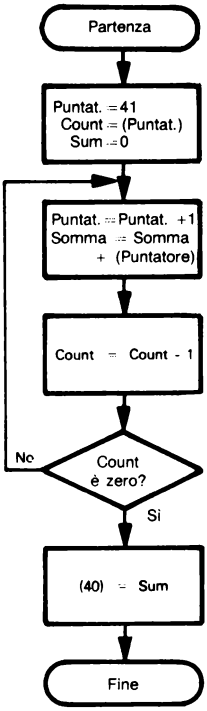
Scopo: Calcolare la somma di una serie di numeri. La lunghezza di questa serie è nella cella di memoria 0041, e la serie comincia nella locazione di memoria 0042. Memorizzare la somma nella cella di memoria 0040. Supporre che la somma sia un numero di 8 bit in modo tale da poter trascurare i riporti.

Problema Campione:

(0041) = 03
(0042) = 28
(0043) = 55
(0044) = 26
Risultato: (0040) = (0042) + (0043) + (0044)
= 28 + 55 + 26
= A3

Ci sono tre ingressi nella somma, dato che (0041) = 03.

Diagramma di Flusso:



Nota: (Puntatore) è il contenuto della cella di memoria indirizzata da Puntatore. Ricordate che sullo Z80, Puntatore è un indirizzo a 16 bit, mentre (Puntatore) è un byte di dati ad 8 bit.

Programma Sorgente:

```
        LD      HL,41H
        LD      B,(HL)      ;COUNT = LUNGHEZZA DELLA SERIE
        SUB     A           ;SUM = ZERO
SUMD:   INC     HL
        ADD     A,(HL)      ;SUM = SUM + DATA
        DEC     B
        JR      NZ,SUMD
        LD      (40H),A     ;IMMAGAZZINA SUM
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)		Istruzione (Codice mnemonico)	
0000	21		LD	HL,41H
0001	41			
0002	00			
0003	46		LD	B,(HL)
0004	97		SUB	A
0005	23	SUMD:	INC	HL
0006	86		ADD	A,(HL)
0007	05		DEC	B
0008	20		JR	NZ,SUMD
0009	FB			
000A	32		LD	(40H),A
000B	40			
000C	00			
000D	76		HALT	

La parte di inizializzazione del programma è costituita dalle prime tre istruzioni che posizionano la somma, il contatore e il puntatore del dato ai loro valori di partenza.

Notate che si può usare LD per trasferire dati fra la memoria e qualunque registro principale di uso generale (cioè A, B, C, E, H, L) utilizzando l'indirizzo nei registri H e L. Comunque, gli unici trasferimenti permessi, se si usa l'indirizzamento diretto, sono quelli che spostano i dati da e verso l'Accumulatore (cioè LD A, (addr) e LD (addr), A — non esiste per esempio l'istruzione LD E, (addr)).

La parte di processo del programma è la sola istruzione ADD A, (HL) che somma il contenuto della locazione di memoria indirizzata dai Registri H e L al contenuto dell'Accumulatore, ed immagazzina il risultato nell'Accumulatore. Questa istruzione esegue il vero e proprio lavoro del programma.

La parte di controllo di ciclo del programma consiste delle istruzioni INC HL e DEC B. INC HL aggiorna il puntatore in modo che la successiva ripetizione sommi il numero susseguente al valore della somma. DEC B decrementa il contatore che non perde di vista il numero di ripetizioni che sono rimaste.

L'istruzione JR NZ porta ad un salto se il Flag Zero vale zero. Lo spostamento (offset) è un numero complemento a due, e il conteggio comincia dalla cella di memoria immediatamente seguente l'istruzione JR. In questo caso, il salto richiesto è dalla locazione di memoria 000A alla locazione 0005. Così lo spostamento è:

$$\begin{array}{r} 0005 \\ -000A \end{array} = \begin{array}{r} 05 \\ \underline{+F6} \\ FB \end{array}$$

Se il flag Zero vale uno, la CPU esegue la successiva istruzione in sequenza (cioè LD (40H), A). Dato che DEC B era l'ultima istruzione prima di JR ad influenzare il flag Zero, JR NZ, SUMD provoca un salto a SUMD se DEC B non produce un risultato pari a zero, cioè:

$$PC = \begin{cases} \text{SUMD se } B \neq 0 \\ PC + 2 \text{ se } B = 0 \end{cases}$$

(Il 2 viene causato dall'istruzione di due byte JR).

La sequenza di controllo di ciclo DEC seguita da JR NZ è così comune che lo Z80 ha un'istruzione speciale che decrementa il contatore e contemporaneamente compie il salto. Questa istruzione è DJNZ. Decrementa e Salta su Non Zero, che decrementa il Registro B e poi salta della quantità relativa specificata se il resto non è pari a zero. Così noi potremmo cambiare la parte finale dell'esempio in:

```
DJNZ    SUMD
LD      (40H), A
HALT
```

Che ha la forma oggetto:

```
07      10      DJNZ    SUMD
08      FC
09      32      LD      (40H), A
0A      40
0B      00
0C      76      HALT
```

Questa variazione fa risparmiare un byte di memoria e tre cicli di clock. Notate, tuttavia, che dovete usare il Registro B come contatore poiché questo è il registro che DJNZ decrementa.

Poiché lo spostamento nei salti relativi dello Z80 è lungo un solo byte, tali salti non possono andare oltre le 127 celle in avanti o le 128 celle all'indietro (in realtà 129 in avanti o 126 all'indietro, dato che il conteggio comincia alla fine dell'istruzione di 2 parole). Salti più ampi devono far uso delle istruzioni JP.

La maggior parte dei cicli del computer eseguono un conteggio verso il basso piuttosto che verso l'alto cosicché il flag Zero può servire quale condizione di uscita. Ricordate che il flag Zero vale 1 se il risultato era zero e vale 0 se il risultato non era zero. Provate a riscrivere il programma in modo che esso conti verso l'alto invece che verso il basso; quale metodo è più efficace?

L'ordine delle istruzioni è spesso molto importante. DEC B deve venire proprio prima di JR NZ, SUMD, poiché altrimenti il risultato di Zero provocato da DEC B potrebbe essere modificato da un'altra istruzione. INC HL deve venire prima di ADD A, (HL) altrimenti il primo numero aggiunto al valore della somma sarà il contenuto della cella di memoria 0041 invece del contenuto della cella 0042.

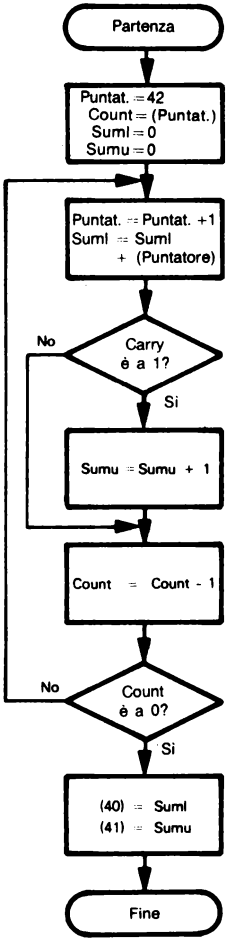
Somma di Dati a 16 Bit

Scopo: Calcolare la somma di una serie di numeri. La lunghezza è nella locazione di memoria 0042 e la serie stessa comincia nella cella 0043. Memorizzare la somma nelle celle 0040 e 0041 (gli otto bit meno significativi in 0040).

Problema Campione:

```
        (0042) = 03
        (0043) = C8
        (0044) = FA
        (0045) = 96
Risultato: C8 + FA + 96 = 0258
        (0040) = 58
        (0041) = 02
```

Diagramma di Flusso:



Programma Sorgente:

```
        LD      HL,42H
        LD      B,(HL)          ;COUNT = LUNGHEZZA DELLA SERIE
        SUB     A               ;I BIT LSB DI SUM = 0
        LD      C,A             ;I BIT MSB DI SUM = 0
DSUMD:  INC     HL
        ADD     A,(HL)          ;SUM = SUM + DATA
        JR      NC,CHCNT
        INC     C               ;SOMMA IL CARRY AI BIT MSB DI SUM
CHCNT:  DJNZ    DSUMD
        LD      HL,40H
        LD      (HL),A          ;IMMAGAZZINA I BIT LSB DI SUM
        INC     HL
        LD      (HL),C          ;IMMAGAZZINA I BIT MSB DI SUM
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)	
0000	21	LD	HL,42H
0001	42		
0002	00		
0003	46	LD	B,(HL)
0004	97	SUB	A
0005	4F	LD	C,A
0006	23	DSUMD: INC	HL
0007	86	ADD	A,(HL)
0008	30	JR	NC.CHCNT
0009	01		
000A	0C	INC	C
000B	10	CHCNT: DJNZ	DSUMD
000C	F9		
000D	21	LD	HL,40H
000E	40		
000F	00		
0010	77	LD	(HL),A
0011	23	INC	HL
0012	71	LD	(HL),C
0013	76	HALT	

La struttura di questo programma è come quella precedentemente considerata. I bit più significativi della somma devono ora essere inizializzati e memorizzati. La parte di processo consiste di tre istruzioni (ADD A, (HL); JR NC, CHCNT; ed INC C) compreso un Salto Condizionato.

JR NC, CHCNT provoca un salto alla locazione di memoria CHCNT se il Carry = 0. Così, se non c'è riporto della somma ad 8 bit, il programma salta intorno allo statement che incrementa i bit più significativi della somma. Lo spostamento relativo è:

```
  000B
 -000A
 ------
    01
```

Il salto relativo per DJNZ DSUMD è:

```
  0006     06
 -000D  =  +F3
          ----
           F9
```

INC somma 1 al contenuto del Registro C. Notate che INC BC è un incremento a 16 bit che somma 1 al Registro C ed aggiunge il riporto al Registro B; INC C è un incremento ad 8 bit che non considera il riporto.

Numero di Elementi Negativi

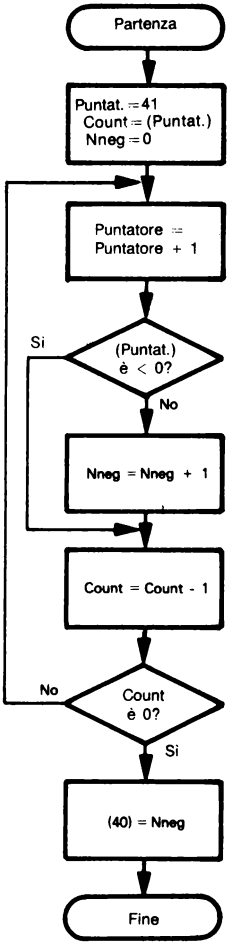
Scopo: Determinare il numero di elementi negativi (con il bit più significativo uguale ad 1) in un blocco. La lunghezza del blocco è nella cella di memoria 0041 e il blocco stesso comincia nella locazione 0042. Porre il numero di elementi negativi nella cella di memoria 0040.

Problema Campione:

(0041) = 06
(0042) = 68
(0043) = F2
(0044) = 87
(0045) = 30
(0046) = 59
(0047) = 2A

Risultato: (0040) = 02, poichè 0043 e 0044 contengono numeri aventi il bit MSB uguale ad 1

Diagramma di Flusso:



Programma Sorgente:

```
        LD      HL,41H
        LD      B,(HL)    ;COUNT = NUMERO DI ELEMENTI
        LD      C,0       ;NUMERO DEI NEGATIVI = ZERO
SRNEG:  INC     HL
        LD      A,(HL)    ;PRENDI L'ELEMENTO SUCCESSIVO
        AND     A         ;IL BIT MSB È ZERO?
        JP      P,CHCNT
        INC     C         ;NO, SOMMA 1 AL NUMERO DEI NEGATIVI
CHCNT:  DJNZ    SRNEG
        LD      A,C       ;IMMAGAZZINA IL NUMERO DEI NEGATIVI
        LD      (40H),A
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)		
0000	21		LD	HL,41H
0001	41			
0002	00			
0003	46		LD	B,(HL)
0004	0E		LD	C,0
0005	00			
0006	23	SRNEG:	INC	HL
0007	7E		LD	A,(HL)
0008	A7		AND	A
0009	F2		JP	P,CHCNT
000A	0D			
000B	00			
000C	0C		INC	C
000D	10	CHCNT:	DJNZ	SRNEG
000E	F7			
000F	79		LD	A,C
0010	32		LD	(40H),A
0011	40			
0012	00			
0013	76		HALT	

AND A posiziona semplicemente i bit di flag a seconda del contenuto dell'Accumulatore senza interferire su quel contenuto; OR A ha l'identico effetto. Ciò è necessario poiché il caricamento dell'Accumulatore non agisce sui flag.

JP P, CHCNT necessita di un indirizzo completo a 16 bit. Non ci sono salti relativi sul flag di Segno come ci sono invece sui flag di Carry e di Zero.

Notate che ciò che noi realmente vogliamo eseguire è il test sul bit 7 della cella di memoria indirizzata dai Registri H e L. Lo Z80 possiede una speciale istruzione, BIT, che analizza il singolo bit e che è specificatamente adibita a questo scopo. Bit posiziona il flag Z con il complemento del bit indicato appartenente al registro o alla cella di memoria indicati. Per esempio BIT S,D posizionerà Z a 1 se il bit 5 del Registro D è zero e la posizionerà a 0 se il bit 5 del registro D è uno. Un'implementazione di questa alternativa è quanto segue.

Programma Sorgente:

```
        LD      HL,41H
        LD      B,(HL)      ;COUNT = NUMERO DI ELEMENTI
        LD      C,0         ;NUMERO DEI NEGATIVI = ZERO
SRNEG:  INC     HL
        BIT     7,(HL)      ;IL PROSSIMO ELEMENTO È NEGATIVO?
        JR      Z,CHCNT
        INC     C           ;SÌ, SOMMA 1 AL NUMERO DEI NEGATIVI
CHCNT:  DJNZ    SRNEG
        LD      A,C         ;IMMAGAZZINA IL NUMERO DEI NEGATIVI
        LD      (40H),A
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)	
0000	21	LD	HL,41H
0001	41		
0002	00		
0003	46	LD	B,(HL)
0004	0E	LD	C,0
0005	00		
0006	23	SRNEG: INC	HL
0007	CB	BIT	7,(HL)
0008	7E		
0009	28	JR	Z,CHCNT
000A	01		
000B	0C	INC	C
000C	10	CHCNT: DJNZ	SRNEG
000D	F8		
000E	79	LD	A,C
000F	32	LD	(40H),A
0010	40		
0011	00		
0012	76	HALT	

BIT 7, (HL) porta ad 1 il bit Z se il bit 7 della locazione di memoria indirizzata dai Registri H e L è zero ed azzera il bit Z se il bit 7 di quella stessa locazione vale uno. BIT non influenza né i registri né le celle di memoria.

Questo programma usa JR Z, CHCNT dato che non sono necessari incrementi se il bit indirizzato è pari a zero.

Ci sarebbe ancora un altro accostamento se si usasse l'istruzione RLC (HL) per spostare il bit di segno del dato di memoria nel Carry. Il salto richiesto sarebbe dunque JR NC, CHCNT. Tuttavia, questo metodo usa del tempo supplementare (RLC (HL) richiede 15 cicli se paragonato ai 12 necessari per BIT 7, (HL) ed inoltre fa variare il dato nella memoria, il quale può essere indispensabile per altri scopi. Notate che questi svantaggi sono messi in relazione; il tempo supplementare è necessario per restituire il risultato alla locazione di memoria.

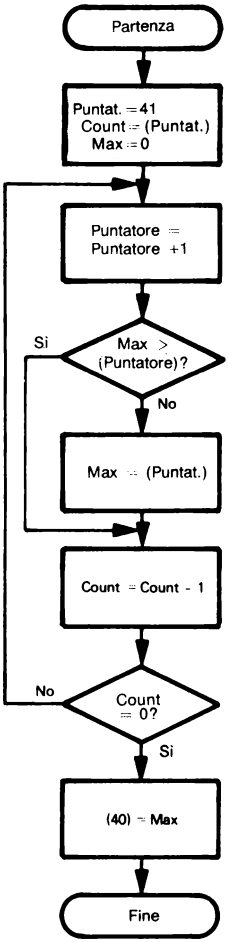
Ricerca del Massimo

Scopo: Trovare l'elemento maggiore in un blocco di dati. La lunghezza di questo blocco è contenuta nella cella di memoria 0041 ed esso stesso inizia nella cella 0042. Memorizzare il massimo nella locazione di memoria 0040. Supporre che i numeri nel blocco siano tutti numeri binari ad 8 bit privi di segno.

Problema Campione:

(0041) = 05
(0042) = 67
(0043) = 79
(0044) = 15
(0045) = E3
(0046) = 72
Risultato: (0040) = E3, poiché questi è il maggiore dei cinque numeri senza segno

Diagramma di flusso:



Programma Sorgente:

```
        LD      HL,41H      ;PUNTA VERSO COUNT
        LD      B,(HL)      ;COUNT = NUMERO DI ELEMENTI
        SUB     A           ;MAXIMUM = VALORE MINIMO POSSIBILE (ZERO)
NEXTE:  INC     HL
        CP      (HL)        ;IL PROSSIMO ELEMENTO È SUPERIORE
                            ;  A MAXIMUM?
        JR      NC,DECNT
        LD      A,(HL)      ;SÌ, SOSTITUISCI L'ELEMENTO IN MAXIMUM
DECNT:  DJNZ    NEXTE
        LD      (40H),A     ;SALVA MAXIMUM
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)		
0000	21		LD	HL,41H
0001	41			
0002	00			
0003	46		LD	B,(HL)
0004	97		SUB	A
0005	23	NEXTE:	INC	HL
0006	BE		CP	(HL)
0007	30		JR	NC,DECNT
0008	01			
0009	7E		LD	A,(HL)
000A	10	DECNT:	DJNZ	NEXTE
000B	F9			
000C	32		LD	(40H),A
000D	40			
000E	00			
000F	76		HALT	

Lo spostamento relativo per JR NC, DECNT è:

```
  000A
 -0009
 ----
    01
```

Lo spostamento relativo per DJNZ NEXTE è:

```
  0005     05
 -000C  =  +F4
 -----     ---
            F9
```

Le prime tre istruzioni di questo programma formano la parte d'inizializzazione.

Questo programma trae profitto dal fatto che zero è il più piccolo numero binario ad 8 bit privo di segno. Quando posizionate il registro che contiene il valore massimo — in questo caso l'Accumulatore — al valore minimo possibile prima di entrare nel ciclo, allora il programma porterà l'Accumulatore al valore più grande a meno che tutti gli elementi nella matrice siano degli zeri.

Il programma lavora correttamente se ci sono due elementi, ma non avviene questo se ve n'è uno solo o nessuno affatto. Perchè? Come potreste risolvere questo problema?

L'istruzione CP (HL) posiziona il flag di Carry come segue (ELEMENT è il contenuto dell'indirizzo formato dai Registri H e L e MAX è il contenuto dell'Accumulatore):

CARRY = 1 se ELEMENT > MAX
CARRY = 0 se ELEMENT ≤ MAX

Se CARRY = 0, il programma prosegue verso DECNT e non fa variare il massimo. Se CARRY = 1, il programma sostituisce il vecchio contenuto massimo con l'elemento corrente eseguendo l'istruzione LD A, (HL).

Il programma non lavora se i numeri sono dotati di segno poiché i numeri negativi appariranno maggiori di quelli positivi. Il problema è piuttosto delicato poiché l'overflow potrebbe far apparire che il risultato ha il segno sbagliato.

Ricordate che l'overflow avviene quando la grandezza di un risultato ha influenza sul suo bit di segno. Lo Z80 ha un flag di Parità/overflow che indica quando è avvenuto un overflow sul complemento a due. Le operazioni aritmetiche che si risolvono in un overflow portano ad 1 questo flag. Potete quindi sondare il suo valore con le istruzioni JP PE, ADDR (Salto su Parità Pari – o Salto su Overflow) oppure JP PO, ADDR (Salto su Parità Dispari – o Salto su Non Overflow). Una cosa che dovreste osservare è che questo impiego dello Z80 è incompatibile con i microprocessori 8080A o 8085. che usano sempre il flag P per indicare la parità. I microprocessori 8080A e 8085 non hanno alcun indicatore di overflow.

Giustificazione di una Frazione Binaria

Scopo: Spostare il contenuto della cella di memoria 0040 verso sinistra finché il bit più significativo del numero sia 1. Memorizzare il risultato nella cella 0041 e il numero degli spostamenti a sinistra richiesti, nella cella di memoria 0042. Se il contenuto della cella 0040 è zero, azzerare sia la cella 0041 che la 0042.

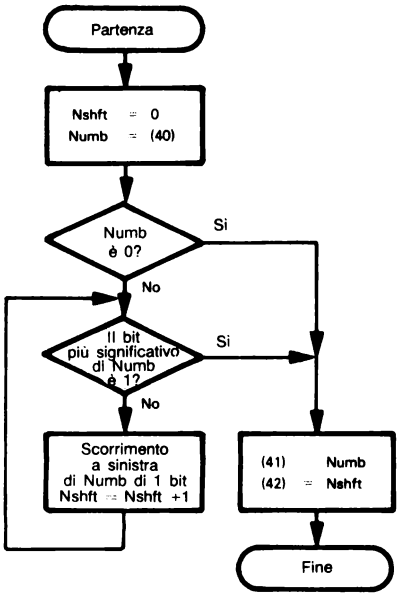
Nota: Il processo è proprio come convertire un numero in una notazione scientifica; per esempio:

$$0{,}0057 = 5{,}7 \times 10^{-3}$$

Problemi Campione:

a.		(0040)	=	22
	Risultato:	(0041)	=	88
		(0042)	=	02
b.		(0040)	=	01
	Risultato:	(0041)	=	80
		(0042)	=	07
c.		(0040)	=	CB
	Risultato:	(0041)	=	CB
		(0042)	=	00
d.		(0040)	=	00
	Risultato:	(0041)	=	00
		(0042)	=	00

Diagramma di Flusso:



Programma Sorgente:

```
        LD      B,0         ;NUMERO DI SCORRIMENTI = ZERO
        LD      HL,40H
        LD      A,(HL)      ;PRENDI IL DATO
        AND     A           ;IL DATO È ZERO?
        JR      Z,DONE      ;SÌ, VAI A DONE
CHKMS:  JP      M,DONE      ;VAI A DONE SE IL BIT DI SEGNO È 1
        INC     B           ;SOMMA 1 AL NUMERO DI SCORRIMENTI
        ADD     A,A         ;FAI SCORRERE A SINISTRA UN BIT
        JP      CHKMS
DONE:   INC     HL
        LD      (HL),A      ;SALVA IL DATO
        INC     HL
        LD      (HL),B      ;SALVA IL NUMERO DI SCORRIMENTI
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)		
0000	06		LD	B,0
0001	00			
0002	21		LD	HL,40H
0003	40			
0004	00			
0005	7E		LD	A,(HL)
0006	A7		AND	A
0007	28		JR	Z,DONE
0008	08			
0009	FA	CHKMS:	JP	M,DONE
000A	11			
000B	00			
000C	04		INC	B
000D	87		ADD	A,A
000E	C3		JP	CHKMS
000F	09			
0010	00			
0011	23	DONE:	INC	HL
0012	77		LD	(HL),A
0013	23		INC	HL
0014	70		LD	(HL),B
0015	76		HALT	

JP M, DONE provoca un salto alla locazione DONE se il bit Segno è pari ad 1. Questa condizione può significare che l'ultimo risultato era un numero negativo o può soltanto significare che il suo bit più significativo era 1 — il computer fornisce solamente il risultato; il programmatore deve provvedere all'interpretazione.

ADD A, A somma il numero nell'Accumulatore a se stesso. Il programma usa questa istruzione, piuttosto di RLA o RLCA, perchè ADD A influenza il bit di Segno diversamente da RLA e RLCA.

Potremmo riorganizzare questo programma in modo da eliminare un JP estraneo ed usare salti relativi anziché assoluti. Una versione riorganizzata sarebbe questa:

```
        LD      B,0         ;NUMERO DI SCORRIMENTI = ZERO
        LD      HL,40H
        LD      A,(HL)      ;PRENDI IL DATO
        AND     A           ;IL DATO È ZERO?
        JR      Z,DONE      ;SÌ, VAI A DONE
        DEC     B           ;DIMINUISCI IL NUMERO DI SCORRIMENTI DI UNO
CHKMS:  INC     B           ;SOMMA 1 AL NUMERO DI SCORRIMENTI
        RLA                 ;FAI SCORRERE A SINISTRA DI UN BIT
        JR      NC,CHKMS    ;CONTINUA SE IL BIT MSB NON È UNO
        RRA                 ;AGGIUSTA INDIETRO IL DATO
DONE:   INC     HL
        LD      (HL),A      ;SALVA IL DATO
        INC     HL
        LD      (HL),D      ;SALVA IL NUMERO DI SCORRIMENTI
        HALT
```

Dimostrate che anche questa versione funziona. Quali sono i vantaggi e svantaggi rispetto al precedente programma?

PROBLEMI

1) Somma di Controllo di Dati (Checksum)

Scopo: Calcolare la somma di controllo (checksum) di una serie di numeri. La lunghezza di questa serie è contenuta nella cella di memoria 0041 ed essa inizia dalla locazione 0042. Memorizzare la somma di controllo nella locazione di memoria 0040. La somma di controllo viene strutturata mediante l'operazione di OR esclusivo su tutti i numeri consecutivamente.

Nota: Tali somme di controllo vengono spesso usate in sistemi a base di nastri di carta e cassette per assicurare che i dati siano stati letti correttamente. La somma di controllo calcolata viene paragonata a quella memorizzata con i dati — se i due controlli non coincidono, il sistema solitamente indicherà un errore all'operatore oppure rileggerà automaticamente il dato.

Problema Campione:

```
            (0041) = 03
            (0042) = 28
            (0043) = 55
            (0044) = 26
Risultato:  (0040) = (0042) ⊕ (0043) ⊕ (0044)
                   = 28 ⊕ 55 ⊕ 26
                   =  00101000
                   ⊕  01010101
                      --------
                      01111101
                   ⊕  00100110
                      --------
                      01011011
                   = 5B
```

2) Somma di Dati a 16 Bit

Scopo: Calcolare la somma di una serie di numeri a 16 bit. La lunghezza della serie è contenuta nelle celle di memoria 0042 e la serie stessa inizia dalla locazione 0043. Memorizzare il risultato della somma nelle celle 0040 e 0041 (gli otto bit più significativi in 0041). Ciascun numero a 16 bit occupa due celle di memoria, con il byte più significativo nell'indirizzo più elevato. Supporre che la somma possa essere contenuta in 16 bit.

Problema Campione:

```
            (0042) = 03
            (0043) = F1
            (0044) = 28
            (0045) = 1A
            (0046) = 30
            (0047) = 89
            (0048) = 4B
Risultato:  28F1 + 301A  +  4B89 = A494
            (0040) = 94
            (0041) = A4
```

3) Numero di Elementi Uguali a Zero, Positivi e Negativi

Scopo: Determinare in un blocco il numero di elementi pari a zero, positivi (il bit più significativo a zero, anche se il numero completo non è zero) e negativi (il bit più significativo ad 1). La lunghezza del blocco è contenuta nella cella di memoria 0043 e il blocco stesso parte dalla locazione 0044. Posizionare il numero di elementi negativi nella cella di memoria 0040, il numero degli elementi uguali a zero nella cella 0041, e il numero di elementi positivi nella cella 0042.

Problema Campione:

(0043) = 06
(0044) = 68
(0045) = F2
(0046) = 87
(0047) = 00
(0048) = 69
(0049) = 2A
Risultato: 2 negativi, 1 zero e 3 positivi, così
(0040) = 02
(0041) = 01
(0042) = 03

4) Ricerca del Minimo

Scopo: Trovare l'elemento minore di un blocco di dati. La lunghezza del blocco è nella locazione di memoria 0041 ed esso inizia dalla cella 0042. Memorizzare il minimo nella cella 0040. Supporre che i numeri compresi nel blocco siano numeri binari ad 8 bit privi di segno.

Problema Campione:

(0041) = 05
(0042) = 67
(0043) = 79
(0044) = 15
(0045) = E3
(0046) = 72
Risultato: (0040) = 15, poiché questi è il minore dei cinque numeri senza segno

5) Conteggio dei Bit Uguali ad 1

Scopo: Determinare quanti bit nella cella di memoria 0040 sono pari a uno e porre il risultato nella locazione 0041.

Problema Campione:

(0040) = 3B = 00111011
Risultato: (0041) = 05

Capitolo 6

DATI CODIFICATI COME CARATTERI

I microprocessori sovente trattano dati codificati come caratteri. Non solo le tastiere, le telescriventi, i dispositivi per le comunicazioni, i visualizzatori ottici e i terminali di computer attendono o forniscono dati codificati in caratteri; anche parecchi strumenti, sistemi di collaudo e di controllo richiedono i dati sotto questa forma. Il codice più comunemente utilizzato è quello ASCII. Più raramente si trovano quelli di Baudot e quello EBCDIC. Supporremo che tutti i nostri dati codificati come caratteri siano ASCII a 7 bit, con il bit più significativo uguale a zero (vedere Tabella 6-1).

Alcuni principi da ricordare nel trattare dati codificati in ASCII sono:

GESTIONE DEI DATI IN ASCII

1) I codici per le cifre e le lettere formano delle successioni ordinate. I codici per le cifre decimali vanno da 30 a 39 in esadecimale, cosicché potete convertire da decimale ad ASCII con un semplice fattore aggiuntivo. I codici per lettere maiuscole sono compresi tra 41 e 54 in esadecimale, in modo tale che potete eseguire la disposizione alfabetica classificando il dato secondo un ordine numerico progressivo.
2) Il computer non fa alcuna distinzione fra caratteri di stampa e non. Questa distinzione è fatta solamente dai dispositivi di I/O.
3) Un dispositivo ASCII tratterà solo caratteri ASCII. Per stampare un 7 su una stampante ASCII, il microprocessore deve inviare 37 esadecimale a quest'ultima; 07 esadecimale è il carattere «bell» (campanello). Ugualmente, il microprocessore riceverà il carattere 9 da una tastiera ASCII come 39 esadecimale; 09 esadecimale è il carattere «tab».
4) Alcuni dispositivi ASCII non usano tutto il blocco di caratteri. Per esempio, i caratteri di controllo e le lettere minuscole possono essere ignorati o stampati come spazi o punti di domanda.
5) Alcuni caratteri ASCII largamente adoperati sono:
   $0A_{16}$ - line feed (LF) (alimentazione diriga)
   $0D_{16}$ - ritorno carrello (CR)
   20_{16} - spazio
   $3F_{16}$ - ? (punto di domanda)
   $7F_{16}$ - caratteri di rubout o delete
6) Ciascun carattere ASCII occupa 7 bit. Ciò permette un vasto insieme di caratteri ma è dispersivo quando i dati sono limitati a una piccola quantità come i numeri decimali. Un Byte di 8 bit, per esempio, può contenere solo un digit decimale codificato in ASCII, mentre ne potrebbe contenere due codificati in BCD.

Tabella 6-1. Tabella Esadecimale ASCII

MSD Esadec. / LSD Esadec.	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	—	o	DEL

ESEMPI

Lunghezza di una Stringa di Caratteri

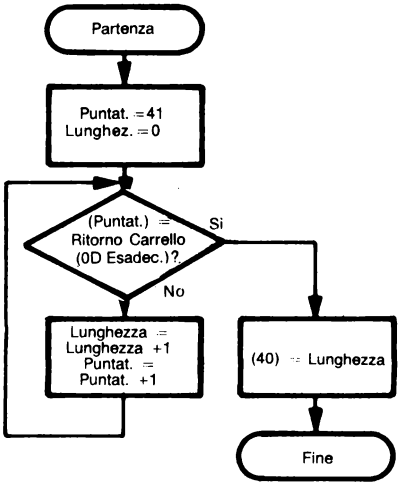
Scopo: Determinare la lunghezza di una stringa di caratteri ASCII (sette bit con il bit più significativo uguale a zero). La stringa inizia nella locazione di memoria 0041; la fine della stringa è contrassegnata da un carattere di ritorno carrello («CR», OD esadecimale). Porre la lunghezza della stringa (escluso il ritorno carrello) nella cella di memoria 0040.

Problemi Campione:

a. (0041) = 0D
Risultato: (0040) = 00 poiché il primo carattere è un ritorno carrello

b. (0041) = 52 'R'
(0042) = 41 'A'
(0043) = 54 'T'
(0044) = 48 'H'
(0045) = 45 'E'
(0046) = 52 'R'
(0047) = 0D CR
Risultato: (0040) = 06

Diagramma di Flusso:



Programma Sorgente:

```
        LD      HL,41H    ;PUNTATORE = INIZIO DI STRINGA
        LD      B,0       ;LUNGHEZZA DI STRINGA = ZERO
        LD      A,0DH     ;PRENDI IL RITORNO CARRELLO IN ASCII
                          ;   PER IL CONFRONTO
CHKCR:  CP      (HL)      ;IL CARATTERE È UN RITORNO CARRELLO?
        JR      Z,DONE    ;SÌ, VAI A DONE
        INC     B         ;NO, SOMMA 1 ALLA LUNGHEZZA DI STRINGA
        INC     HL
        JR      CHKCR     ;PROVA COL PROSSIMO CARATTERE
DONE:   LD      A,B       ;SALVA LA LUNGHEZZA DI STRINGA
        LD      (40H),A
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)		
0000	21		LD	HL,41H
0001	41			
0002	00			
0003	06		LD	B,0
0004	00			
0005	3E		LD	A,0DH
0006	0D			
0007	BE	CHKCR:	CP	(HL)
0008	28		JR	Z,DONE
0009	04			
000A	04		INC	B
000B	23		INC	HL
000C	18		JR	CHKCR
000D	F9			
000E	78	DONE:	LD	A,B
000F	32		LD	(40H),A
0010	40			
0011	00			
0012	76		HALT	

Il ritorno carrello (CR) è esattamente un altro carattere ASCII (OD esadecimale) per quanto riguarda il computer. Il fatto che il dispositivo di uscita tratti il ritorno carrello come un carattere di controllo piuttosto che come un carattere da stampare non riguarda il computer.

L'istruzione di confronto, CP, posiziona i flag come se fosse stata eseguita una sottrazione, ma lascia il carattere di ritorno carrello nell'Accumulatore per i successivi confronti. Il flag di Zero (Z) viene influenzato come segue:

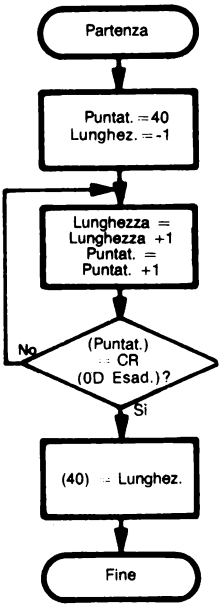
Z = 1 se il carattere nella stringa è un ritorno carrello
Z = 0 se non c'è un ritorno carrello

L'istruzione INC B somma 1 al contatore della lunghezza della stringa nel Registro B. LD B, 0 inizializza questo contatore a zero prima dell'inizio del ciclo. Ricordate di inizializzare le variabili prima di usarle in un ciclo.

Questo ciclo non termina poiché un contatore viene decrementato fino a zero. Il computer continuerà semplicemente ad esaminare caratteri finché trova un ritorno carrello. Potete disporre un conteggio massimo in un ciclo come questo per evitare problemi con le stringhe errate che non contengono un ritorno carrello. Cosa avverrebbe se il programma nell'esempio fosse usato con una stringa di questo tipo?

Notate che, riordinando la logica e modificando le condizioni iniziali, potete abbreviare il programma e diminuire il suo tempo di esecuzione. Se noi correggiamo il diagramma di flusso in modo che il programma incrementi il contatore e il puntatore prima di cercare il carattere di ritorno carrello, si rende necessaria una sola istruzione di Jump invece di due. Il nuovo diagramma di flusso e il programma sono i seguenti:

Diagramma di flusso:



Programma Sorgente:

```
        LD      HL,40H          ;PUNTATORE = BYTE PRIMA DI STRINGA
        LD      B,0FFH          ;LUNGHEZZA = -1
        LD      A,0DH           ;PRENDI IL RITORNO CARRELLO IN ASCII PER
                                ;   IL CONFRONTO
CHKCR:  INC     HL
        INC     B               ;SOMMA 1 A LUNGHEZZA DI STRINGA
        CP      (HL)            ;IL CARATTERE È UN RITORNO CARRELLO?
        JR      NZ,CHKCR        ;NO, CONTROLA IL PROSSIMO CARATTERE
        LD      A,B             ;SÌ, SALVA LUNGHEZZA DI STRINGA
        LD      (40H),A
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)		
0000	21		LD	HL,40H
0001	40			
0002	00			
0003	06		LD	B,0FFH
0004	FF			
0005	3E		LD	A,0DH
0006	0D			
0007	23	CHKCR:	INC	HL
0008	04		INC	B
0009	BE		CP	(HL)
000A	20		JR	NZ,CHKCR
000B	FB			
000C	78		LD	A,B
000D	32		LD	(40H),A
000E	40			
000F	00			
0010	76		HALT	

È molto comune dover ricercare valori particolari in una lista, tabella, o stringa. Il microprocessore Z80 ha, infatti, delle istruzioni speciali che semplificano questo compito.

Queste particolari istruzioni sono chiamate Istruzioni di Esplorazione Blocco; esse funzionano in questo modo:

ISTRUZIONI DI ESPLORAZIONE BLOCCO

CPI confronta il contenuto della cella di memoria indirizzata da HL con il contenuto dell'Accumulatore (esattamente come CP (HL)). Quindi incrementa HL e decrementa il contatore di byte (coppia di registri BC). Il bit di Parità/Overflow viene azzerato se il contatore di byte raggiunge lo zero dopo il decremento e in caso contrario è al valore 1. CPD è la stessa istruzione con la sola differenza che decrementa HL anziché incrementarlo.

CPIR e CPDR sono le forme ripetute delle Istruzioni di Esplorazione Blocco. Queste istruzioni ripetono l'istruzione fondamentale di Esplorazione finché decrementando BC non si raggiunge zero oppure non avviene un confronto positivo (cioè, A = (HL)). Ricordate che il raggiungimento del valore zero in seguito al decremento di BC porta all'azzeramento del bit di Parità/Overflow, mentre la scoperta di una uguaglianza porta ad 1 il bit di Zero.

Notate che BC contiene un contatore a 16 bit. Così, le Istruzioni di Esplorazione Blocco possono manipolare stringhe di qualsiasi lunghezza.

Una versione del precedente programma che utilizza CPI è riportata di seguito.

Programma Sorgente:

```
        LD      HL,41H      ;PUNTATORE = INIZIO DI STRINGA
        LD      BC,0        ;CONTATORE DEI BYTE = ZERO
        LD      A,0DH       ;PRENDI IL RITORNO CARRELLO IN CODICE
                            ;  ASCII PER IL CONFRONTO
CHKCR:  CPI                 ;IL CARATTERE È UN RITORNO CARRELLO?
        JR      NZ,CHKCR    ;NO, CONTROLLA IL PROSSIMO CARATTERE
        LD      A,0FFH      ;SÌ, CALCOLA LUNGHEZZA DI STRINGA
        SUB     C
        LD      (40H),A     ;SALVA LUNGHEZZA DI STRINGA
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)		
0000	21		LD	HL,41H
0001	41			
0002	00			
0003	01		LD	BC,0
0004	00			
0005	00			
0006	3E		LD	A,0DH
0007	0D			
0008	ED	CHKCR:	CPI	
0009	A1			
000A	20		JR	NZ,CHKCR
000B	FC			
000C	3E		LD	A,0FFH
000D	FF			
000E	91		SUB	C
000F	32		LD	(40H),A
0010	40			
0011	00			
0012	76		HALT	

È necessaria una piccola manipolazione per calcolare la lunghezza della stringa, dato che CPI decrementa il contatore di byte (BC) anziché incrementarlo come è stato fatto con INC B nel precedente programma. Inoltre, il contatore di byte viene decrementato un'ulteriore volta quando viene trovato il ritorno carrello. Come potremmo adattare le condizioni iniziali per trattare questo problema?

In realtà, possiamo migliorare il programma, ancor di più usando CPIR per eliminare la necessità del salto relativo JR. CPIR esegue le stesse cose di CPI, ed inoltre ripete automaticamente la procedura di confronto a meno che A = (HL) o BC sia giunto a zero decrementandosi. Il programma che usa CPIR è riportato di seguito.

Programma Sorgente:

```
LD      HL,41H   ;PUNTATORE = INIZIO DI STRINGA
LD      BC,0     ;CONTATORE DEI BYTE = ZERO
LD      A,0DH    ;PRENDI IL RITORNO CARRELLO IN ASCII PER
                 ;  IL CONFRONTO
CPIR             ;RICERCA DI UN RITORNO CARRELLO
LD      A,0FFH   ;CALCOLA LA LUNGHEZZA DI STRINGA
                 ;  DA CONTATORE
SUB     C
LD      (40H),A  ;SALVA LUNGHEZZA DI STRINGA
HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)	
0000	21	LD	HL,41H
0001	41		
0002	00		
0003	01	LD	BC,0
0004	00		
0005	00		
0006	3E	LD	A,0DH
0007	0D		
0008	ED	CPIR	
0009	B1		
000A	3E	LD	A,0FFH
000B	FF		
000C	91	SUB	C
000D	32	LD	(40H),A
000E	40		
000F	00		
0010	76	HALT	

Le Istruzioni con funzione multipla come CPI e CPIR hanno lo stesso effetto delle sequenze che esse sostituiscono. I risparmi nel tempo di esecuzione e nella memoria si ottengono perchè il processore necessita di meno istruzioni per ciascun passaggio attraverso il ciclo. Pertanto, il vero guadagno si ha nell'esecuzione del ciclo.

In tutti questi programmi si suppone che la stringa sia lunga meno di 256 byte. Come li modifichereste per trattare stringhe più lunghe?

Ricerca del Primo Carattere Diverso da Blank

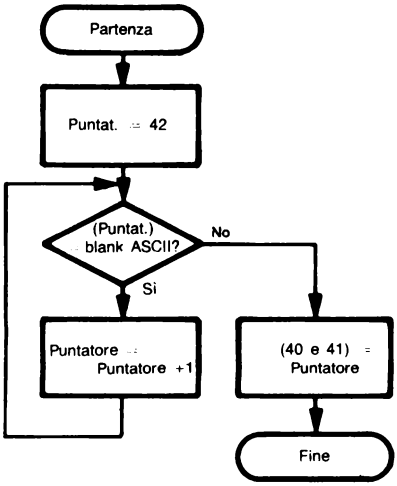
Scopo: Cercare in una stringa di caratteri ASCII (sette bit con il bit più significativo uguale a zero) un carattere diverso da blank (spazio). La stringa inizia nella cella di memoria 0042. Porre l'indirizzo del primo carattere diverso da blank nelle locazioni di memoria 0040 e 0041 (i bit più significativi in 0041). Un carattere di blank è pari a 20 esadecimale in codice ASCII.

Problemi Campione:

a. (0042) = 37 '7'
Risultato: (0040) = 42, poiché la locazione di memoria 0042 contiene un carattere non-blank
(0041) = 00

b. (0042) = 20 SP
(0043) = 20 SP
(0044) = 20 SP
(0045) = 46 F
(0046) = 20 SP
Risultato: (0040) = 45, poiché le tre precedenti locazioni di memoria contengono tutte blank
(0041) = 00

Diagramma di flusso:



Programma sorgente:

```
        LD      HL,42H      ;PUNTATORE = INIZIO DI STRINGA
        LD      A,20H       ;PRENDI IL CODICE ASCII DI BLANK PER IL
                            ;   CONFRONTO
CHBLK:  CP      (HL)        ;IL CARATTERE ASCII È UN BLANK?
        JR      NZ,DONE     ;NO, VAI AVANTI
        INC     HL
        JR      CHBLK       ;SÌ, ESAMINA IL CARATTERE SUCCESSIVO
DONE:   LD      (40H),HL    ;NO, SALVA L'INDIRIZZO DEL PRIMO CARATTERE
                            ;   NON BLANK
        HALT
```

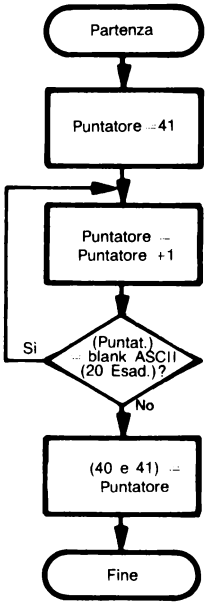
Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)		
0000	21		LD	HL,42H
0001	42			
0002	00			
0003	3E		LD	A,20H
0004	20			
0005	BE	CHBLK:	CP	(HL)
0006	20		JR	NZ,DONE
0007	03			
0008	23		INC	HL
0009	18		JR	CHBLK
000A	FA			
000B	22	DONE:	LD	(40H),HL
000C	40			
000D	00			
000E	76		HALT	

È molto comune dover ricercare spazi nelle stringhe. Gli spazi vengono spesso eliminati dalle stringhe quando vengono usati semplicemente per aumentare la leggibilità oppure per ottenere particolari formati. È ovviamente controproducente memorizzare e trasmettere degli spazi d'inizio, di fine od altri supplementari, particolarmente se torna svantaggioso dal punto di vista della capacità dei mezzi di comunicazione e della memoria richiesta. Il caricamento di dati e di programmi, tuttavia, è molto più semplice se sono ammessi degli spazi in più. I microcomputer sono sovente utilizzati per casi come questo per convertire dati fra forme che sono facili da usare per l'uomo e forme che sono trattate efficientemente sui computer e sulle linee dei mezzi di comunicazione.

L'istruzione LD (addr), HL è conveniente per memorizzare indirizzi nel formato Z80 (prima il byte meno significativo). LD (40H), HL immagazzina il contenuto del Registro L nella cella di memoria 0040 e il contenuto del Registro H nella cella 0041.

Inoltre, se alteriamo le condizioni iniziali in modo che la parte di controllo ciclo preceda la parte di processo, possiamo ridurre il numero di byte del programma e diminuire il tempo di esecuzione del ciclo. Il diagramma di flusso riordinato è:



Programma Sorgente:

```
        LD      HL,41H   ;PUNTA VERSO IL BYTE PRIMA DELLA STRINGA
        LD      A,20H    ;PRENDI IL CODICE ASCII DI BLANK PER
                         ;  IL CONFRONTO
CHBLK:  INC     HL
        CP      (HL)     ;IL CARATTERE ASCII È UN BLANK?
        JR      Z,CHBLK  ;SÌ, CONTINUA AD ESAMINARE CARATTERI
        LD      (40H),HL ;NO, SALVA L'INDIRIZZO DEL PRIMO CARATTERE
                         ;NON-BLANK
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)		
0000	21		LD	HL,41H
0001	41			
0002	00			
0003	3E		LD	A,20H
0004	20			
0005	23	CHBLK:	INC	HL
0006	BE		CP	(HL)
0007	28		JR	Z,CHBLK
0008	FC			
0009	22		LD	(40H),HL
000A	40			
000B	00			
000C	76		HALT	

Come nell'esempio precedente, potremmo sostituire la sequenza INC HL, CP (HL) con la sola istruzione CPI. Comunque, poiché non abbiamo bisogno del contatore di byte in questo programma, CPI necessita della stessa quantità di memoria (2 byte) e di più tempo (16 cicli di clock anziché 13) delle istruzioni che essa sostituisce. Non potremmo qui utilizzare CPIR dato che si vuole che il programma termini quando i caratteri **non** sono gli stessi.

Sostituzione degli Zeri di Testa con Caratteri di Blank

Scopo: Redarre l'editing di una stringa di caratteri decimali in codice ASCII sostituendo tutti gli zeri di testa con dei caratteri di blank. La stringa inizia nella cella di memoria 0041; supporre che questa consista interamente di cifre decimali codificate in ASCII. La lunghezza della stringa è contenuta nella cella di memoria 0040.

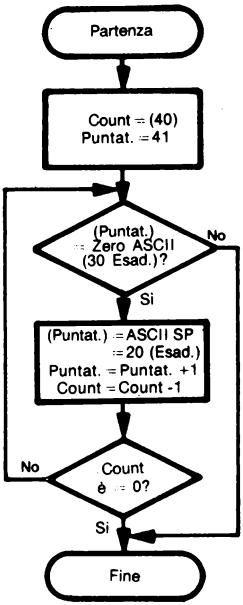
Problemi Campione:

a.
(0040) = 02
(0041) = 36 '6'

Il programma lascia immutata la stringa, poiché il digit di guida non è zero.

b.
(0040) = 08
(0041) = 30 '0'
(0042) = 30 '0'
(0043) = 38 '8'
Risultato: (0041) = 20 SP
(0042) = 20 SP

Diagramma di flusso:



Programma Sorgente:

```
        LD      HL,40H
        LD      B,(HL)    ;COUNT = LUNGHEZZA DI STRINGA
        LD      A,'0'     ;PRENDI ZERO IN ASCII PER IL CONFRONTO
CHKZ:   INC     HL
        CP      (HL)      ;IL DIGIT DI GUIDA È ZERO?
        JR      NZ,DONE   ;NO, VAI AVANTI
        LD      (HL),20H  ;SOSTITUISCI IL DIGIT DI GUIDA CON BLANK
        DJNZ    CHKZ      ;ESAMINA IL DIGIT SUCCESSIVO, SE C'È
DONE:   HALT
```

Apici singoli tra caratteri indicano un codice ASCII.

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)		Istruzione (Codice mnemonico)	
0000	21		LD	HL,40H
0001	40			
0002	00			
0003	46		LD	B,(HL)
0004	3E		LD	A,'0'
0005	30			
0006	23	CHKZ:	INC	HL
0007	BE		CP	(HL)
0008	20		JR	NZ,DONE
0009	04			
000A	36		LD	(HL),20H
000B	20			
000C	10		DJNZ	CHKZ
000D	F8			
000E	76	DONE:	HALT	

Si vuole sovente redarre l'editing di stringhe decimali prima che vengano stampate o visualizzate per perfezionare la loro apparenza. I compiti comuni dell'editing comprendono l'eliminazione degli zeri di testa, dei numeri di riconoscimento, dei segni di somma o altri segni di riconoscimento e degli arrotondamenti. Chiaramente, dei numeri stampati come 0006 oppure $27,34382 possono confondere o dar noia.

Qui il ciclo ha due uscite — una se il processore trova una cifra che non è zero e l'altra se ha esaminato la stringa completa.

L'istruzione LD (HL), 204 pone 20 (esadecimale) nella locazione di memoria indirizzata dai Registri H e L. Potreste anche inizializzare il Registro C con 20H (cioè, LD C, 204) e usare LD (HL), C per sostituire lo zero di testa con un carattere di blank. Notate in questo esempio i vantaggi conseguenti. LD (HL), C viene eseguita più rapidamente di LD (HL), 20H e diminuisce il tempo di esecuzione all'interno del ciclo. Tuttavia è richiesta una istruzione LD C, 20H nella parte inizializzazione della routine. Se questo esempio dovesse essere usato in un'applicazione come registratore di cassa, quale sequenza sceglieresteе e perchè?

Tutte le cifre nella stringa si suppongono in codice ASCII; vale a dire, le cifre sono esadecimali da 30 a 39, anziché le normali decimali da 0 a 9. La conversione da decimale ad ASCII si risolve semplicemente sommando 30 esadecimale alla cifra decimale.

Bisogna fare in modo, quando si sostituiscono gli zeri leading (iniziali) con un carattere di blank, di lasciare uno zero nel caso che tutte le cifre siano pari a zero. Come fareste questo?

Notate che ciascuna cifra in ASCII necessita di otto bit, in confronto ai quattro per una cifra espressa in BCD. Perciò l'ASCII è un formato dispendioso se si vogliono memorizzare o trasmettere dati numerici.

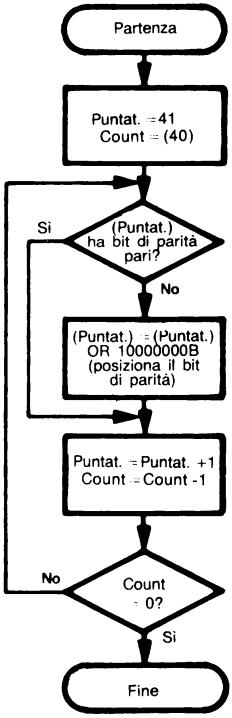
Aggiunta della Parità di Tipo Pari a Caratteri ASCII

Scopo: Aggiungere la parità di tipo pari ad una stringa di caratteri ASCII a 7 bit. La lunghezza della stringa è contenuta nella cella di memoria 0040 e la stringa stessa inizia nella locazione 0041. Porre la parità di tipo pari nel bit più significativo di ciascun carattere portando il bit più significativo ad 1 se ciò fa in modo che il numero totale di bit uguali ad 1 nella parola sia un numero pari.

Problema Campione:

	(0040)	=	06
	(0041)	=	31
	(0042)	=	32
	(0043)	=	33
	(0044)	=	34
	(0045)	=	35
	(0046)	=	36
Risultato:	(0041)	=	B1
	(0042)	=	B2
	(0043)	=	33
	(0044)	=	B4
	(0045)	=	35
	(0046)	=	36

Diagramma di Flusso:



Programma Sorgente:

```
        LD      HL,40H
        LD      B,(HL)          ;PRENDI LUNGHEZZA DI STRINGA
        LD      C,10000000B     ;PRENDI IL BIT DI PARITÀ DI 1
SETPR:  INC     HL
        LD      A,(HL)          ;PRENDI UN CARATTERE
        OR      C               ;POSIZIONA AD 1 IL BIT DI PARITÀ
                                ;  E CONTROLLA LA PARITÀ
        JP      PO,CHCNT        ;LA PARITÀ È ORA PARI?
        LD      (HL),A          ;SÌ, SALVA IL CARATTERE CON PARITÀ PARI
CHCNT:  DJNZ    SETPR
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)		
0000	21		LD	HL,40H
0001	40			
0002	00			
0003	46		LD	B,(HL)
0004	0E		LD	C,10000000H
0005	80			
0006	23	SETPR:	INC	HL
0007	7E		LD	A,(HL)
0008	B1		OR	C
0009	E2		JP	PO,CHCNT
000A	0D			
000B	00			
000C	77		LD	(HL),A
000D	10	CHCNT:	DJNZ	SETPR
000E	F7			
000F	76		HALT	

La parità viene sovente aggiunta ai caratteri ASCII prima che questi vengano trasmessi su linee di comunicazione perturbate, per fornire una semplice forma di controllo degli errori. La parità rivela tutti gli errori su un solo bit ma non permette la loro correzione (cioè, si può sapere che è avvenuto un errore quando la parità ricevuta è sbagliata, ma non si può comunicare quale bità è variato).
LD C, 10000000B conserva un bit di parità di 1 nel Registro C. (Notate l'uso della maschera binaria; lo scopo della maschera è più chiaro quando essa viene specificata in questo modo anziché 80H o 128 decimale).
L'istruzione ORC porta il bit di parità (più significativo) ad 1 mentre conserva tutti gli altri bit com'erano e porta pure il flag di Parità dello Z80 ad 1.
La seguente procedura viene usata per determinare se la parità del byte di memoria è di tipo dispari oppure pari. Si esegue una operazione logica di OR sul bit di parità nel byte caricato dalla memoria e quindi lo si analizza per vedere se si tratta di parità di tipo dispari. Se la parità è di tipo dispari, allora il byte nella memoria ha una parità di tipo pari, e si salta verso il basso a decrementare il conteggio dei byte rimanenti. Se la parità è di tipo pari, allora si sa che il byte nella memoria ha parità di tipo dispari, e perciò si memorizza il byte contenuto nell'Accumulatore in quella locazione di memoria.
I salti condizionati JP PO (Salto da Parità Dispari) e JP PE (Salto da Parità Pari) vengono usati raramente tranne che nella generazione e nel controllo di parità. Notate che non esistono salti **relativi** condizionati dal valore del bit di Parità, come non ce ne sono pure condizionati dal valore del bit di Segno.

Non confondete il bit di Parità incluso in ciascun carattere ed il flag di Parità dello Z80, che viene posizionato ad 1 se l'ultimo risultato aritmetico o Booleano ha una parità di tipo pari.

Un metodo alternativo utilizza la serie delle istruzioni dello Z80. Questa versione necessita di maggior tempo ma non richiede l'uso di un registro temporaneo per il bit di parità.

Programma Sorgente:

```
        LD      HL,40H
        LD      B,(HL)      ;PRENDI LUNGHEZZA DI STRINGA
SETPR:  INC     HL
        LD      A,(HL)      ;PRENDI UN CARATTERE
        OR      A           ;IL CARATTERE HA PARITÀ PARI?
        JP      PE,CHCNT
        SET     7,(HL)      ;NO, POSIZIONA IL BIT DI PARITÀ AD 1
CHCNT:  DJNZ    SETPR
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)		
0000	21		LD	HL,40H
0001	40			
0002	00			
0003	46		LD	B,(HL)
0004	23	SETPR:	INC	HL
0005	7E		LD	A,(HL)
0006	B7		OR	A
0007	EA		JP	PE,CHCNT
0008	0C			
0009	00			
000A	CB		SET	7,(HL)
000B	FE			
000C	10	CHCNT:	DJNZ	SETPR
000D	F6			
000E	76		HALT	

Confronto con Stringa Campione

Scopo: Confrontare due stringhe di caratteri ASCII per vedere se sono gli stessi. La lunghezza delle stringhe è contenuta nella cella di memoria 0041: una stringa inizia nella cella 0042 e l'altra nella cella 0052. Se le due stringhe sono identiche, azzerare la cella di memoria 0040, altrimenti, posizionare la cella di memoria 0040 al valore FF esadecimale (tutti i bit ad 1).

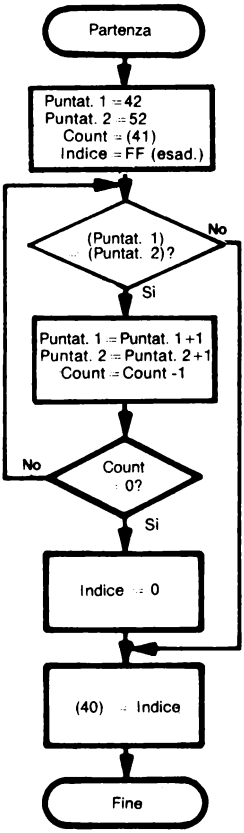
Problemi Campione:

a.
```
        (0040) = 03
        (0042) = 43  'C'
        (0043) = 41  'A'
        (0044) = 54  'T'
        (0052) = 43  'C'
        (0053) = 41  'A'
        (0054) = 54  'T'
Risultato: (0040) = 00,  poiché le due stringhe sono le stesse.
```

b. (0041) = 03
(0042) = 52 'R'
(0043) = 41 'A'
(0044) = 54 'T'
(0052) = 43 'C'
(0053) = 41 'A'
(0054) = 54 'T'
Risultato: (0040) = FF, poiché i primi caratteri nelle stringhe differiscono.

Nota: Il processo di verifica d'identicità termina appena la CPU trova una divergenza – non è necessario che venga esaminata la parte restante della stringa.

Diagramma di flusso:



Programma Sorgente:

```
        LD      HL,41H
        LD      B,(HL)      ;COUNT = LUNGHEZZA DI STRINGHE
        INC     HL          ;PUNTATORE 1 = INIZIO DI STRINGA 1
        LD      DE,52H      ;PUNTATORE 2 = INIZIO DI STRINGA 2
        LD      C,0FFH      ;INDICE = FF (ESADECIMALE)
CHCAR:  LD      A,(DE)      ;PRENDI UN CARATTERE DALLA STRINGA 2
        CP      (HL)        ;C'È UNA CORRISPONDENZA?
        JR      NZ,DONE     ;NO, VAI A DONE
        INC     DE
        INC     HL
        DJNZ    CHCAR       ;CONTROLLA LA COPPIA SUCCESSIVA SE
                            ;  CE N'È ANCORA
        LD      C,0         ;INDICE = 0 SE TUTTI I CARATTERI
                            ;  CONCORDANO
DONE:   LD      A,C
        LD      (40H),A     ;SALVA L'INDICE
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)		
0000	21		LD	HL,41H
0001	41			
0002	00			
0003	46		LD	B,(HL)
0004	23		INC	HL
0005	11		LD	DE,52H
0006	52			
0007	00			
0008	0E		LD	C,0FFH
0009	FF			
000A	1A	CHCAR:	LD	A,(DE)
000B	BE		CP	(HL)
000C	20		JR	NZ,DONE
000D	06			
000E	13		INC	DE
000F	23		INC	HL
0010	10		DJNZ	CHCAR
0011	F8			
0012	0E		LD	C,0
0013	00			
0014	79	DONE:	LD	A,C
0015	32		LD	(40H),A
0016	40			
0017	00			
0018	76		HALT	

La verifica d'uguaglianza di stringhe di caratteri ASCII è una parte essenziale nella ricerca dei comandi, nomi di riconoscimento, variabili di identificazione o codici di operazione negli assemblatori e compilatori, ritrovamento di file e molti altri compiti.

Il programma fa uso di due puntatori, uno contenuto nella Coppia di Registri HL e l'altro nella Coppia di Registri DE. Le sole istruzioni che usano l'indirizzo in DE sono LD A, (DE) (Caricamento dell'Accumulatore dalla Locazione di Memoria Indirizzata da DE) ed LD (DE), A (Me-

morizzazione dell'Accumulatore nella Locazione di Memoria Indirizzata da DE). Le operazioni Logiche ed Aritmetiche con la memoria ed i trasferimenti da e verso altri registri (ad es. ADD A, (HL); AND (HL); LD B, (HL); LD (HL), E) possono essere eseguite usando esclusivamente l'indirizzo nella Coppia di Registri HL, o mediante un registro indice.

L'ordine delle operazioni è molto importante a causa del ridotto numero d'istruzioni che usano l'indirizzo nella Coppia di Registri DE. Si deve spostare un carattere dalla stringa individuata da DE verso l'Accumulatore e confrontarlo a un carattere della stringa puntata da HL. Quest'ordine di operazioni è necessario dato che lo Z80 non ha istruzioni che permettono il confronto con un carattere in una stringa puntata da DE.

Per esempio, se sostituiste LD A, (DE) con LD A, (HL) quale sarebbe la successiva istruzione? Questa asimmetria è tipica dello Z80 e può provocare seri problemi nella programmazione.

Notate che ciascuna ripetizione aggiorna entrambi i puntatori.

Questo programma potrebbe avvantaggiarsi del fatto che è noto che un registro contiene il valore zero dopo che viene eseguito un particolare salto condizionato. Quando si esegue l'istruzione DJNZ CHCAR, se non si è compiuto il salto, allora sappiamo che il Registro B contiene zero.

Perciò possiamo spostare il contenuto del Registro B nel Registro C, il nostro registro segnalatore, per indicare che è stata verificata una eguaglianza.

Potremmo anche usare le istruzioni di SET e RESET dello Z80 per manipolare il flag se avessimo la necessità di conservare dei bit per altri motivi.

PROBLEMI

1) Lunghezza di un Messaggio per Telescrivente

Scopo: Determinare la lunghezza di un messaggio ASCII. Tutti i caratteri sono in codice ASCII a 7 bit con MSB = 0. La stringa di caratteri nella quale il messaggio viene caricato inizia dalla cella di memoria 0041. Il messaggio stesso comincia con un carattere ASCII STX (02 esadecimale) e termina con ETX (03 esadecimale). Porre la lunghezza del messaggio (il numero di caratteri compresi tra STX e ETX escluso questi due) nella cella di memoria 0040.

Problema Campione:

```
(0041) = 40
(0042) = 02  STX
(0043) = 47  'G'
(0044) = 4F  'O'
(0045) = 03  ETX
Risultato: (0040) = 02,  poiché ci sono due caratteri tra STX
                         in locazione 0042 ed ETX in locazione 0045.
```

2) Ricerca dell'Ultimo Carattere Diverso da Blank

Scopo: Esplorare una stringa di caratteri ASCII alla ricerca dell'ultimo carattere diverso da blank. La stringa inizia nella cella di memoria 0042 e termina con un carattere di ritorno carrello (OD esadecimale). Porre l'indirizzo dell'ultimo carattere diverso da blank nelle locazioni di memoria 0040 e 0041 (i bit più significativi in 0041).

Problemi Campione:

a. (0042) = 37 '7'
(0043) = 0D CR
Risultato: (0040) = 42, poiché l'ultimo (e l'unico) carattere non-blank è nella locazione di memoria 0042.
(0041) = 00

b. (0042) = 41 'A'
(0043) = 20 SP
(0044) = 48 'H'
(0045) = 41 'A'
(0046) = 54 'T'
(0047) = 20 SP
(0048) = 20 SP
(0049) = 0D CR
Risultato: (0040) = 46
(0041) = 00

3) Riduzione di Stringa Decimale alla Sola Parte Intera

Scopo: Redarre l'editing di una stringa di caratteri ASCII decimali sostituendo tutte le cifre alla destra del punto decimale con caratteri ASCII di blank (20 esadecimale). La stringa inizia nella cella di memoria 0041 e si suppone formata completamente da cifre decimali codificate in ASCII e di un possibile punto decimale (2E esadecimale). La lunghezza della stringa è contenuta nella cella di memoria 0040. Se non appare nessun punto decimale nella stringa, supporre che esso sia implicitamente all'estrema destra.

Problemi Campione:

a. (0040) = 04
(0041) = 37 '7'
(0042) = 2E '.'
(0043) = 38 '8'
(0044) = 31 '1'
Risultato: (0041) = 37 '7'
(0042) = 2E '.'
(0043) = 20 SP
(0044) = 20 SP

b. (0040) = 03
(0041) = 26 '6'
(0042) = 37 '7'
(0043) = 31 '1'
Risultato: Immutato, così il numero si suppone pari a 671.

4) Controllo di Parità di Tipo Pari nei Caratteri ASCII

Scopo: Controllare la parità di tipo pari in una stringa di caratteri ASCII. La lunghezza della stringa è contenuta nella locazione di memoria 0041 e la stringa stessa inizia dalla locazione 0042. Se la parità di tutti i caratteri nella stringa è corretta, azzerare la cella di memoria 0040; altrimenti porre il valore FF esadecimale (tutti i bit ad 1) nella cella 0040.

Problemi campione:

a.
(0041) = 03
(0042) = B1
(0043) = B2
(0044) = 33
Risultato: (0040) = 00, poiché tutti i caratteri hanno parità pari.

b.
(0041) = 03
(0042) = B1
(0043) = B6
(0044) = 33
Risultato: (0040) = FF, poichè il carattere in locazione di memoria 0042 non ha parità pari.

5) Confronto fra Stringhe

Scopo: Confrontare due stringhe di caratteri ASCII per vedere quale è la maggiore (cioè, quale segue l'altra in ordine «alfabetico»). La lunghezza delle stringhe è contenuta nella cella 0041; una stringa inizia dalla cella 0042 e l'altra dalla cella 0052. Se la stringa che inizia dalla cella 0042 è maggiore od uguale all'altra, azzerare la locazione di memoria 0040; altrimenti, posizionare quest'ultima con FF esadecimale (tutti i bit ad 1).

Problemi Campione:

a.
(0041) = 03
(0042) = 43 'C'
(0043) = 41 'A'
(0044) = 54 'T'
(0052) = 42 'B'
(0053) = 41 'A'
(0054) = 54 'T'
Risultato: (0040) = 00, poiché CAT è maggiore di BAT.

b.
(0041) = 03
(0042) = 43 'C'
(0043) = 41 'A'
(0044) = 54 'T'
(0052) = 43 'C'
(0053) = 41 'A'
(0054) = 54 'T'
Risultato: (0040) = 00, poiché le due stringhe sono uguali.

c.
(0041) = 03
(0042) = 43 'C'
(0043) = 41 'A'
(0044) = 54 'T'
(0052) = 43 'C'
(0053) = 55 'U'
(0054) = 54 'T'
Risultato: (0040) = FF, poiché CUT è maggiore di CAT.

Capitolo 7
CONVERSIONE DI CODICE

La conversione di codice è un continuo problema nella maggior parte delle applicazioni del microcomputer. I dispositivi periferici forniscono dati in codice ASCII, BCD od altri codici speciali. Il sistema deve convertire i dati per l'elaborazione in una certa forma standard. I dispositivi di uscita possono richiedere dati codificati in ASCII, BCD, sette segmenti o in altri codici. Perciò, il sistema deve convertire i risultati in una forma opportuna dopo che viene eseguita l'elaborazione.

Vi sono diversi modi per eseguire una conversione di codice:

1) Alcune conversioni possono essere facilmente trattate mediante algoritmi che comportano l'uso di funzioni aritmetiche e logiche. Il programma può, tuttavia, dover trattare alcuni casi speciali separatamente.

2) Conversioni più complesse possono essere trattate mediante tabelle di riferimento. Questo metodo richiede una piccola programmazione ed è semplice da applicare. Comunque, la tabella può occupare una grande quantità di memoria se il campo di variazione dei valori d'ingresso è ampio.

3) L'hardware è disponibile facilmente per alcune funzioni di conversione. Esempi tipici sono i decodificatori per la conversione da BCD a sette segmenti ed i Trasmettitori/Ricevitori Asincroni Universali (UART) per la conversione tra formati parallelo (ASCII) e seriali (telescrivente).

Nella maggior parte delle applicazioni, il programma dovrebbe eseguire il maggior lavoro possibile per la conversione di codice. Ciò produce un risparmio nei componenti e nello spazio della piastra oltre che una maggiore affidabilità. Inoltre, la maggior parte delle conversioni di codice sono facili da programmare e richiedono uno scarso tempo di esecuzione.

ESEMPI

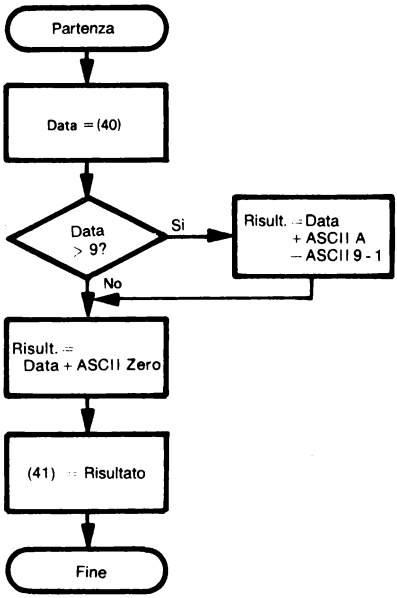
Da Esadecimale ad ASCII

Scopo: Convertire il contenuto della cella di memoria 0040 in un carattere ASCII. La locazione di memoria 040 contiene una sola cifra esadecimale (i quattro bit più significativi sono zero). Memorizzare il carattere ASCII nella locazione 0041.

Problemi Campione:

a.		(0040)	=	0C
	Risultato:	(0041)	=	43 'C'
b.		(0040)	=	06
	Risultato:	(0041)	=	36 '6'

Diagramma di Flusso:



Programma Sorgente:

```
        LD      A,(40H)   ;PRENDI IL DATO
        CP      10        ;IL DATO È 10 O MAGGIORE?
        JR      C,ASCZ
        ADD     A,'A'-'9'-1 ;SÌ, SOMMA L'OFFSET PER LE LETTERE
ASCZ:   ADD     A,'0'     ;SOMMA L'OFFSET PER IL CODICE ASCII
        LD      (41H),A   ;IMMAGAZZINA IL RISULTATO IN ASCII
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)	
0000	3A	LD	A,(40H)
0001	40		
0002	00		
0003	FE	CP	10
0004	0A		
0005	38	JR	C,ASCZ
0006	02		
0007	C6	ADD	A,'A'-'9'-1
0008	07		
0009	C6	ASCZ: ADD	A,'0'
000A	30		
000B	32	LD	(41H),A
000C	41		
000D	00		
000E	76	HALT	

In questo programma, l'idea di base è sommare lo 0 ASCII a tutte le cifre esadecimali. Questa addizione converte in modo corretto le cifre decimali; tuttavia, c'è un'interruzione fra 9 in ASCII (39 esadecimale) che deve essere presa in considerazione. Questa interruzione deve essere aggiunta alle cifre non decimali A, B, C, D, E ed F. Ciò si realizza con l'istruzione ADD A che somma l'offset (spostamento) 'A' - '9' - 1 al contenuto dell'Accumulatore. Potete spiegare perché lo spostamento è 'A' - '9' - 1?

Notate che i termini dell'addizione sono posti nel programma in linguaggio assembly sotto la forma ASCII (un carattere o una stringa di caratteri in ASCII sono racchiusi tra due apostrofi). L'offset per le lettere viene lasciato sotto forma di espressione aritmetica. Lo sforzo è quello di rendere i termini nel listing in linguaggio assembly i più chiari possibile. Il tempo assembly supplementare è un prezzo molto piccolo da pagare per ottenere un notevole aumento della chiarezza.

Questa routine potrebbe essere usata in una varietà di programmi; per esempio, i programmi monitor debbono convertire le cifre esadecimali in ASCII allo scopo di visualizzare il contenuto delle celle di memoria in esadecimale su una stampante o un terminale video che funzionano in ASCII.

Un altro metodo di conversione (più veloce) che non necessita affatto di alcun salto condizionale, è il seguente programma, descritto da Allison nella rivista Computer[1].

```
LD      A,(40H)   ;PRENDI IL DIGIT ESADECIMALE
ADD     A,90H     ;SVILUPPA UN 6 SUPPLEMENTARE E IL CARRY
DAA
ADC     A,40H     ;SOMMA IN CARRY L'OFFSET ASCII
DAA
LD      (41H),A   ;IMMAGAZZINA IL DIGIT IN ASCII
HALT
```

Provate questo programma su qualche cifra. Potete spiegare perché funziona?

Da Decimale a Sette Segmenti

Scopo: Convertire il contenuto della locazione di memoria 0040 in un codice a sette segmenti e porlo nella locazione 0042. Se la locazione 0040 non contiene una sola cifra decimale, azzerare la cella di memoria 0042.

Tabella dei sette segmenti: La seguente tabella può essere usata per convertire numeri decimali in codice a sette segmenti. Il codice a sette segmenti è organizzato con il bit più significativo sempre a zero seguito dal codice (1 = on (acceso), 0 = off (spento) per i segmenti g, f, e, d, c, b, ed a (vedi Figura 7-1).

Digit	Codice
0	3F
1	06
2	5B
3	4F
4	66
5	6D
6	7D
7	07
8	7F
9	6F

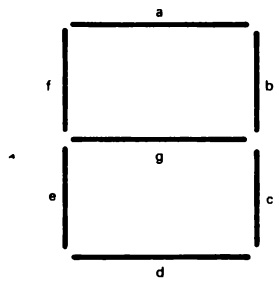


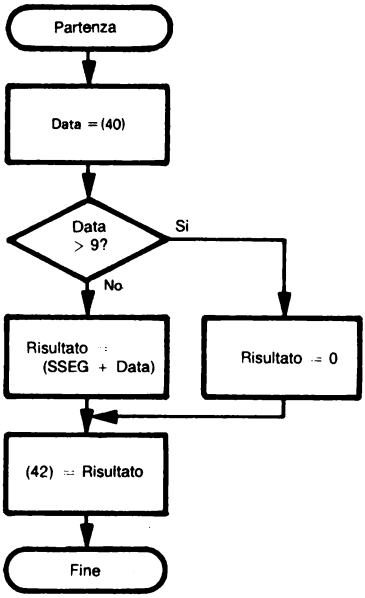
Figura 7-1 Combinazione dei sette segmenti

Notate che la tabella usa 7D per la cifra 6 anziché 7C (con il segmento più alto spento) per evitare di fare confusione con la lettera b minuscola, e 6F per la cifra 9 invece di 67 (con il segmento più basso spento), per nessuna ragione particolare.

Problemi Campione:

a. (0040) = 03
Risultato: (0042) = 4F
b. (0040) = 28
Risultato: (0042) = 00

Diagramma di Flusso:



Notate che la somma dell'indirizzo di base SSEG con l'indice produce l'indirizzo che contiene la risposta.

Programma Sorgente:

```
        LD      B,0         ;PRENDI IL CODICE ERRORE PER SPEGNERE
                            ;   IL DISPLAY
        LD      A,(40H)     ;PRENDI IL DATO
        CP      10          ;IL DATO È UN DIGIT DECIMALE?
        JR      NC,DONE     ;NO, CONSERVA IL CODICE DI ERRORE
        LD      L,A         ;SÌ, ATTRIBUISCI IL DATO AD UN INDICE
                            ;   A 16 BIT
        LD      H,0
        LD      DE,SSEG     ;PRENDI L'INDIRIZZO DI BASE DELLA TABELLA
                            ;   A 7 SEGMENTI
        ADD     HL,DE       ;TROVA L'ELEMENTO MEDIANTE INDICIZZAZIONE
        LD      B,(HL)      ;PRENDI IL CODICE A 7 SEGMENTI DALLA TABELLA
DONE:   LD      A,B         ;SALVA IL CODICE A 7 SEGMENTI O CODICE
                            ;   DI ERRORE
        LD      (42H),A
        HALT
        ORG     20H         ;TABELLA DEI CODICI A SETTE SEGMENTI
SSEG:   DEFB    3FH
        DEFB    06H
        DEFB    5BH
        DEFB    4FH
        DEFB    66H
        DEFB    6DH
        DEFB    7DH
        DEFB    07H
        DEFB    7FH
        DEFB    6FH
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)		
0000	06		LD	B,0
0001	00			
0002	3A		LD	A,(40H)
0003	40			
0004	00			
0005	FE		CP	10
0006	0A			
0007	30		JR	NC,DONE
0008	08			
0009	6F		LD	L,A
000A	26		LD	H,0
000B	00			
000C	11		LD	DE,SSEG
000D	20			
000E	00			
000F	19		ADD	HL,DE
0010	46		LD	B,(HL)
0011	78	DONE:	LD	A,B
0012	32		LD	(42H),A
0013	42			
0014	00			
0015	76		HALT	
0020	3F	SSEG:	DEFB	3FH
0021	06		DEFB	06H
0022	5B		DEFB	5BH
0023	4F		DEFB	4FH
0024	66		DEFB	66H
0025	6D		DEFB	6DH
0026	7D		DEFB	7DH
0027	07		DEFB	07H
0028	7F		DEFB	7FH
0029	6F		DEFB	6FH

Il programma calcola l'indirizzo di memoria del codice desiderato sommando l'indice (cioè la cifra da visualizzare) all'indirizzo base della tabella di codice a sette segmenti. Questa procedura è nota come tabella di consultazione.

La pseudo-operazione in linguaggio assembly DEFB (Define Byte) pone il dato costante nella memoria programmi. Tali dati possono comprendere tabelle, intestazioni, messaggi d'errore, messaggi riempitivi, caratteri di formato, caratteri di riferimento, ecc. Alla label attribuita alla pseudo-operazione DEFB viene assegnato il valore dell'indirizzo nel quale il byte del dato viene calcolato.

Le tabelle sono spesso usate per eseguire conversioni di codice che sono più complesse dell'esempio precedente. Queste tabelle contengono tipicamente tutti i risultati organizzati secondo i dati d'ingresso, ad esempio il primo ingresso è il codice corrispondente al numero zero.

I visualizzatori a sette segmenti forniscono forme riconoscibili di cifre decimali, di alcuni lettere ed altri caratteri. i visualizzatori a sette segmenti usati nei computer sono poco costosi, facili da combinare e consumano poca potenza. Tuttavia, le cifre codificate a sette segmenti sono piuttosto difficili da leggere.

L'assemblatore pone semplicemente il dato per la tabella nella memoria. Notate che una pseudo-operazione DEFB occupa un byte di memoria. Abbiamo lasciato dello spazio in memoria tra il programma e la tabella per permettere le somme e le correzioni successive.

USO DEI REGISTRI INDICE DELLO Z80

Un metodo alternativo sarebbe l'uso di uno dei registri indice dello Z80, diciamo IX. Il programmatore deve tenere presenti le seguenti caratteristiche dei registri indice dello Z80:

1) Lo spostamento fisso nella memoria programmi è di soli otto bit e pertanto non può contenere un indirizzo completo di memoria. Esso deve essere usato o come breve spostamento o per contenere gli otto bit meno significativi di un indirizzo di memoria.
2) I registri indice sono a 16 bit. IX o IY possono essere caricati dalla memoria proprio come una coppia di registri — da due indirizzi consecutivi di memoria con gli otto bit meno significativi in corrispondenza dell'indirizzo più basso.
3) Tutte le operazioni che comportano l'uso dei registri indice hanno bisogno di tempo e memoria supplementari poiché una parola del codice operativo indica semplicemente che sta per essere utilizzato un registro indice.

Il seguente programma utilizza il Registro IX per eseguire la tabella di riferimento:

Programma Sorgente:

```
        LD      B,0             ;PRENDI IL CODICE DI ERRORE PER
                                ;   SPEGNERE IL DISPLAY
        LD      A,(40H)         ;PRENDI IL DATO
        CP      10              ;IL DATO È UN DIGIT DECIMALE?
        JR      NC,DONE         ;NO, CONSERVA IL CODICE DI ERRORE
        LD      HL,41H          ;SALVA IL NUMERO DELLA PAGINA
                                ;   DI TABELLA IN MEMORIA
        LD      (HL),0
        LD      IX,(40H)        ;PRENDI L'OFFSET DI TABELLA
        LD      B,(IX+SSEG)     ;PRENDI IL CODICE A SETTE SEGMENTI
                                ;   DALLA TABELLA
DONE:   LD      A,B             ;SALVA IL CODICE A SETTE SEGMENTI O IL
                                ;   CODICE DI ERRORE
        LD      (42H),A
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)		
0000	06		LD	B,0
0001	00			
0002	3A		LD	A,(40H)
0003	40			
0004	00			
0005	FE		CP	10
0006	0A			
0007	30		JR	NC,DONE
0008	0C			
0009	21		LD	HL,41H
000A	41			
000B	00			
000C	36		LD	(HL),0
000D	00			
000E	DD		LD	IX,(40H)
000F	2A			
0010	40			
0011	00			
0012	DD		LD	B,(IX+SSEG)
0013	46			
0014	20			
0015	78	DONE:	LD	A,B
0016	32		LD	(42H),A
0017	42			
0018	00			
0019	76		HALT	

L'istruzione di caricamento indicizzato LD B, (IX + SSEG) somma l'indice (cioè la cifra da visualizzare) alla base della tabella a sette segmenti per ottenere l'indirizzo del codice desiderato. Notate che il registro indice a 16 bit contiene il dato nei suoi otto bit meno significativi e l'indirizzo di partenza della tabella nei suoi otto bit più significativi. Questa strana combinazione è necessaria perchè lo spostamento incluso nell'istruzione indicizzata è di soli otto bit e può pertanto contenere solamente gli otto bit meno significativi dell'indirizzo di partenza della tabella.

Un programma più generale potrebbe prevedere il posizionamento della tabella in una qualunque parte della memoria. Se l'indirizzo di partenza della tabella è SSEGM (gli otto MSB) e SSEGL (gli otto LSB), l'istruzione LD (HL), 0 deve essere sostituito da LD (HL), SSEGM. Perchè è necessaria questa variazione?

Notate che tutte le operazioni che coinvolgono il Registro Indice IX hanno un codice operativo di 2 parole delle quali la prima è DD.

SPOSTAMENTO DI DATI IN UN BLOCCO

Chiaramente questo non è un uso efficiente dei registri indice. Questi registri diventano veramente utili allorché si devono accedere parecchi dati in un blocco. Il blocco potrebbe contenere le caratteristiche di un messaggio, i parametri di una equazione, lo stato corrente di un processo o di una macchina, o i dati per un visualizzatore video. Potreste, per esempio, prendere il contenuto della dodicesima locazione nel blocco e spostarla nella ventesima locazione mediante entrambe i programmi seguenti, supponendo che l'indirizzo di partenza del blocco sia memorizzato nelle locazioni di memoria PTR e PTR+1.

1) Utilizzando DE ed HL

```
LD      DE,(PTR)      ;PRENDI L'INDIRIZZO DI PARTENZA
LD      HL,12         ;CALCOLA L'INDIRIZZO DELLA SORGENTE
ADD     HL,DE
LD      A,(HL)        ;PRENDI IL DATO DALLA SORGENTE
LD      HL,20         ;CALCOLA L'INDIRIZZO DI DESTINAZIONE
ADD     HL,DE
LD      (HL),A        ;SPOSTA IL DATO NELLA DESTINAZIONE
```

2) Utilizzando IX

```
LD      IX,(PTR)      ;PRENDI L'INDIRIZZO DI PARTENZA
LD      A,(IX+12)     ;PRENDI IL DATO DALLA SORGENTE
LD      (IX+20),A     ;SPOSTA IL DATO NELLA DESTINAZIONE
```

Il programma che fa uso dei registri indice è di gran lunga più corto e più chiaro. La sua unica limitazione è che lo spostamento deve essere sufficientemente piccolo da poter essere contenuto in un byte ad 8 bit.

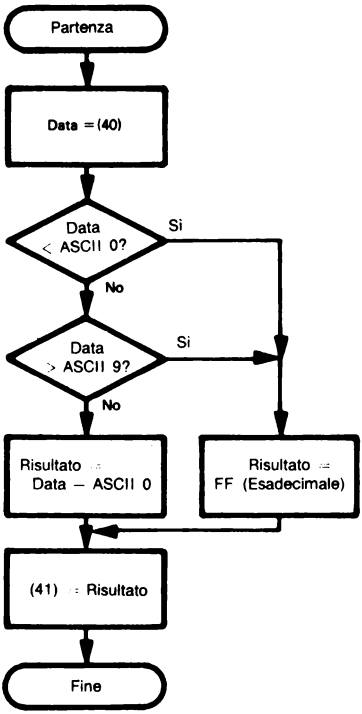
Da ASCII a Decimale

Scopo: Convertire il contenuto della cella di memoria 0040 da un carattere ASCII ad una cifra decimale e memorizzare il risultato nella cella di memoria 0041. Se il contenuto della cella 0040 non è la rappresentazione di una cifra decimale, posizionare il contenuto della iocazione di memoria 0041 ad FF (esadecimale).

Problemi Campione:

a. (0040) = 37 '7'
Risultato: (0041) = 07
b. (0040) = 55
Risultato: (0041) = FF

Diagramma di Flusso:



Programma Sorgente:

```
        LD      B,0FFH      ;PRENDI L'INDICATORE DI ERRORE
        LD      A,(40H)     ;PRENDI IL DATO
        SUB     '0'         ;IL DATO È INFERIORE ALL'ASCII ZERO?
        JR      C,DONE      ;SÌ, NON È UN DIGIT
        CP      '9'+1       ;IL DATO È SUPERIORE ALL'ASCII NOVE?
        JR      NC,DONE     ;SÌ, NON È UN DIGIT
        LD      B,A         ;SALVA IL DIGIT SE È VALIDO
DONE:   LD      A,B         ;SALVA IL DIGIT O L'INDICATORE DI ERRORE
        LD      (41H),A
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)		
0000	06		LD	B,OFFH
0001	FF			
0002	3A		LD	A,(40H)
0003	40			
0004	00			
0005	D6		SUB	'0'
0006	30			
0007	38		JR	C,DONE
0008	05			
0009	FE		CP	'9'+1
000A	3A			
000B	30		JR	NC,DONE
000C	01			
000D	47		LD	B,A
000E	78	DONE:	LD	A,B
000F	32		LD	(41H),A
0010	41			
0011	00			
0012	76		HALT	

Questo programma tratta i caratteri codificati in ASCII esattamente come numeri ordinari. Notate che le cifre decimali e le lettere formano gruppi di codici consecutivi. Stringhe di lettere (come ad esempio nomi) possono essere alfabetizzate ponendo le loro rappresentazioni ASCII in ordine numerico crescente (ASCII B = ASCII A + 1 per esempio).

Sottraendo lo Zero ASCII (30 esadecimale) da una qualsiasi cifra decimale in ASCII si ottiene la rappresentazione BCD di quella cifra.

La conversione da ASCII a decimale è necessaria quando dei numeri decimali stanno per essere incamerati da un dispositivo ASCII come una telescrivente o un terminale video.

L'idea fondamentale del programma è quella di determinare se il carattere è compreso tra lo 0 ASCII e il 9 ASCII incluso. Se ciò avviene, il carattere è una cifra decimale in ASCII, dal momento che le cifre formano una squenza. Esso può quindi essere convertito al valore decimale semplicemente sottraendo 30 esadecimale (0 ASCII), ad es. 7 ASCII - 0 ASCII = 37 - 30 = 7.

Notate che viene fatto un confronto con il risultato di una vera e propria sottrazione (SUB '0') dal momento che questa è necessaria per convertire da ASCII a decimale. L'altro confronto viene eseguito con una sottrazione implicita (CP '9' + 1) dato che il risultato finale è a questo punto contenuto nell'Accumulatore se il numero originale era valido.

Da BCD a Binario

Scopo: Convertire due cifre BCD contenute nelle celle di memoria 0040 e 0041 in un numero binario contenuto nella cella di memoria 0042. La cifra BCD significativa è nella cella 0040.

Problemi Campione:

a. (0040) = 02
(0041) = 09
Risultato: (0042) = 1D (esadecimale) = 29 (decimale)

b. (0040) = 07
(0041) = 01
Risultato: (0042) = 47 (esadecimale) = 71 (decimale)

Nota: Non viene riportato alcun diagramma di flusso dal momento che il programma moltiplica la cifra più significativa per 10 usando semplicemente la formula 10 X = 8 X + 2X. La moltiplicazione per 2 richiede uno spostamento aritmetico verso sinistra e la moltiplicazione per 8 necessita di tre di tali spostamenti.

Programma Sorgente:

```
LD      HL,40H   ;PRENDI IL DIGIT PIÙ SIGNIFICATIVO (MSD)
LD      A,(HL)
ADD     A,A      ;MSD MOLTIPLICATO DUE
LD      B,A      ;SALVA MSD MOLTIPLICATO DUE
ADD     A,A      ;MSD MOLTIPLICATO QUATTRO
ADD     A,A      ;MSD MOLTIPLICATO OTTO
ADD     A,B      ;MSD MOLTIPLICATO DIECI
INC     HL       ;PUNTA AL DIGIT MENO SIGNIFICATIVO
ADD     A,(HL)   ;ADDIZIONA PER OTTENERE L'EQUIVALENTE
                 ;  BINARIO
INC     HL
LD      (HL),A   ;IMMAGAZZINA L'EQUIVALENTE BINARIO
HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)	
0000	21	LD	HL,40H
0001	40		
0002	00		
0003	7E	LD	A,(HL)
0004	87	ADD	A,A
0005	47	LD	B,A
0006	87	ADD	A,A
0007	87	ADD	A,A
0008	80	ADD	A,B
0009	23	INC	HL
000A	86	ADD	A,(HL)
000B	23	INC	HL
000C	77	LD	(HL),A
000D	76	HALT	

Gli ingressi BCD vengono convertiti in binario onde risparmiare sulla memorizzazione e semplificare i calcoli. Tuttavia, la conversione può controbilanciare alcuni degli svantaggi della memorizzazione e della aritmetica binaria.

Questo programma moltiplica la cifra BCD nella locazione di memoria 0040 per dieci usando somme ripetute[2]. Notate che ADD A, A moltiplica il contenuto dell'Accumulatore per 2. Questo vi permette di moltiplicare il contenuto dell'Accumulatore per dei numeri decimali piccoli con poche istruzioni. Come usereste questa procedura per moltiplicare per 16? Per 12? Per 7?

I numeri BCD richiedono circa il 20% in più di memoria rispetto ai numeri binari. La rappresentazione da 0 a 999 necessita di 12 bit in forma BCD ma solamente di 10 bit in forma binaria (dato che $2^{10} = 1024 \simeq 1000$).

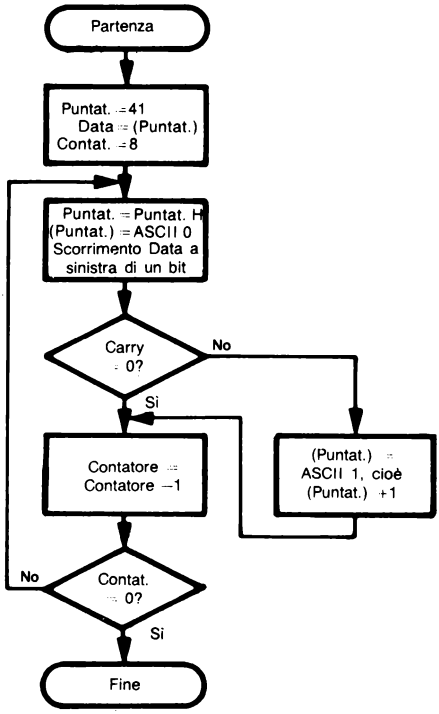
Conversione di un Numero Binario in una Stringa ASCII

Scopo: Convertire il numero binario ad 8 bit contenuto nella cella 0041 in otto caratteri ASCII (ASCII 0 oppure ASCII 1) contenuti nelle celle di memoria da 0042 a 0049 (il bit più significativo in 0042).

Problema Campione:

	(0041)	=	D2 = 11010010	
Risultato:	(0042)	=	31	'1'
	(0043)	=	31	'1'
	(0044)	=	30	'0'
	(0045)	=	31	'1'
	(0046)	=	30	'0'
	(0047)	=	30	'0'
	(0048)	=	31	'1'
	(0049)	=	30	'0'

Diagramma di Flusso:



Programma Sorgente:

```
        LD      HL,41H
        LD      A,(HL)          ;PRENDI IL DATO
        LD      B,8             ;CONTATORE = NUMERO DI BIT NELLA PAROLA
        LD      C,'0'           ;PRENDI IL CODICE ASCII ZERO PER
                                ;  IMMAGAZZINARLO NELLA STRINGA
CONV:   INC     HL
        LD      (HL),C          ;METTI IL CODICE ASCII ZERO NELLA STRINGA
        RLA                     ;IL PROSSIMO BIT DEL DATO È 1?
        JR      NC,COUNT        ;SÌ, METTI AD 1 IN ASCII L'ELEMENTO DI STRINGA
        INC     (HL)
COUNT:  DJNZ    CONV
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)		
0000	21		LD	HL,41H
0001	41			
0002	00			
0003	7E		LD	A,(HL)
0004	06		LD	B,8
0005	08			
0006	0E		LD	C,'0'
0007	30			
0008	23	CONV:	INC	HL
0009	71		LD	(HL),C
000A	17		RLA	
000B	30		JR	NC,COUNT
000C	01			
000D	34		INC	(HL)
000E	10	COUNT:	DJNZ	CONV
000F	F8			
0010	76		HALT	

Le cifre ASCII formano una sequenza in modo che ASCII 1 = ASCII 0 + 1. Ricordate che i registri dello Z80 hanno usi particolari. Dovreste porre il contatore di ciclo nel Registro B in modo da poter usare l'istruzione DJNZ.

È necessario badare alla differenza fra INC HL, che somma uno al contenuto a 16 bit della Coppia di Registri HL, e INC (HL), che somma uno al contenuto ad 8 bit della locazione di memoria indirizzata dalla Coppia di Registri HL.

La conversione da Binario ad ASCII si rende necessaria allorquando dei numeri vengono stampati in forma binaria su un dispositivo ASCII.

La conversione da ASCII comporta semplicemente la somma di ASCII 0 (30 esadecimale).

PROBLEMI

1) Da ASCII ad Esadecimale

Scopo: Convertire il contenuto della cella di memoria 0040 in una cifra esadecimale e memorizzare il risultato nella cella 0041. Supporre che la cella di memoria 0040 contenga la rappresentazione ASCII di una cifra esadecimale (7 bit con MSB pari a 0).

Problemi Campione:

a. (0040) = 43 'C'
Risultato: (0041) = 0C
b. (0040) = 36 '6'
Risultato: (0041) = 06

2) Da Sette-Segmenti a Decimale

Scopo: Convertire il contenuto della locazione di memoria 0040 dal codice a sette-segmenti ad un numero decimale contenuto nella cella 0041. Se la locazione di memoria 0040 no contiene un codice a sette segmenti valido, allora posizionare la cella di memoria 0041 al valore FF (esadecimale). Usare la tabella dei sette segmenti fornita di seguito all'esempio di conversione «Da Decimale a 7 Segmenti» e provare ad abbinare i codici.

Problemi Campione:

a. (0040) = 4F
Risultato: (0041) = 03
b. (0040) = 28
Risultato: (0041) = FF

3) Da Decimale ad ASCII

Scopo: Convertire il contenuto della cella di memoria 0040 da una cifra decimale a carattere ASCII e memorizzare il risultato nella cella 0041. Se il numero nella cella di memoria 0040 non è una cifra decimale, posizionare il contenuto della locazione 0041 ad un valore pari al carattere ASCII di blank (20 esadecimale).

Problemi Campione:

a. (0040) = 07
Risultato: (0041) = 37 '7'
b. (0040) = 55
Risultato: (0041) = 20 SP

4) Da Binario a BCD

Scopo: Convertire il contenuto della locazione di memoria 0040 in due cifre BCD contenute nelle celle di memoria 0041 e 0042 (la cifra più significativa in 0041). Il numero nella cella 0040 è privo di segno e minore di 100.

Problemi Campione:

a.		(0040)	= 1D	(29 decimale)
	Risultato:	(0041)	= 02	
		(0042)	= 09	
b.		(0040)	= 47	(71 decimale)
	Risultato:	(0041)	= 07	
		(0042)	= 01	

5) Da Stringa ASCII a Numero Binario

Scopo: Convertire gli otto caratteri ASCII contenuti nelle celle di memoria da 0042 a 0049 in un numero binario ad 8 bit contenuto nella cella 0041 (il bit più significativo in 0042). Azzerare la cella di memoria 0040 se tutti i caratteri ASCII sono ASCII 1 oppure ASCII 0, altrimenti posizionarla ad FF.

Problemi Campione:

a.		(0042)	= 31	'1'
		(0043)	= 31	'1'
		(0044)	= 30	'0'
		(0045)	= 31	'1'
		(0046)	= 30	'0'
		(0047)	= 30	'0'
		(0048)	= 31	'1'
		(0049)	= 30	'0'
	Risultato:	(0041)	= D2	
		(0040)	= 00	
b. come 'a' tranne che:				
		(0045)	= 37	'7'
	Risultato:	(0040)	= FF	

BIBLIOGRAFIA

1. Allison, D.R. "A Design Philosophy for Microcomputer Architectures", Computer, Febbraio 1977, pagg. 35-41. Questo è un eccellente articolo che raccomandiamo vivamente.
2. Altri metodi di conversione da BCD a binario vengono trattati da J.A. Tabb e M.L. Roginsky, "Microprocessor Algorithms Make BCD-Binary Conversions Super-fast", EDN, 5 Gennaio 1977, pagg. 46-50 e da J.B. Peatman, Microcomputer-based Design, Mc Graw-Hill, New York, 1977, pagg. 400-406.

Capitolo 8

PROBLEMI ARITMETICI

La maggior parte delle operazioni aritmetiche in applicazioni che usano il microprocessore consiste di manipolazioni su parole multiple binarie e decimali. Una correzione decimale (adattamento decimale) e qualche altro mezzo per eseguire l'aritmetica decimale è spesso la sola istruzione fornita oltre a quelle fondamentali di somma e sottrazione. Dovete implementare le altre operazioni aritmetiche con sequenze di istruzioni.

L'aritmetica binaria in precisione multipla richiede delle semplici ripetizioni delle istruzioni di base a singolo byte. Il bit di Carry trasferisce l'informazione fra parole. La Somma con Carry e la Sottrazione con Carry utilizzano l'informazione proveniente dalle operazioni aritmetiche precedenti. Si deve badare ad azzerare il Carry prima di operare sulla prima parola (chiaramente non c'è nessun riporto nei prestititi dai bit meno significativi).

L'aritmetica decimale è un compito abbastanza comune per i microprocessori che per la maggior parte posseggono istruzioni speciali adatte a questo scopo. Queste istruzioni possono eseguire operazioni decimali direttamente oppure correggere il risultato di operazioni binarie nella giusta forma decimale. L'aritmetica decimale è essenziale in applicazioni quali terminali per punti di vendita, calcolatori, processori di controllo, sistemi d'entrata ordine, e terminali di banca.

Potete implementare la moltiplicazione e la divisione come serie di addizioni e sottrazioni rispettivamente, proprio come esse possano essere eseguite a mano. Le operazioni a due byte sono necessarie dato che una moltiplicazione produce un risultato due volte più lungo degli operandi, mentre una divisione riduce analogamente la lunghezza del risultato. Le moltiplicazioni e le divisioni sprecano del tempo se sono eseguite in software a causa delle ripetute operazioni aritmetiche e di spostamento che si rendono necessarie. Naturalmente, moltiplicare o dividere per una potenza di 2 è semplice poiché tali operazioni possono essere implementate mediante un opportuno numero di spostamenti aritmetici verso sinistra o verso destra.

ESEMPI

Addizione in Multi-precisione

Scopo: Sommare due numeri binari multi-byte. La lunghezza dei numeri binari (in byte) è contenuta nella locazione di memoria 0040, mentre i numeri stessi iniziano (con il bit meno significativo) dalle celle di memoria 0041 e 0051, rispettivamente, e la somma sostituisce il numero che inizia nella cella di memoria 0041.

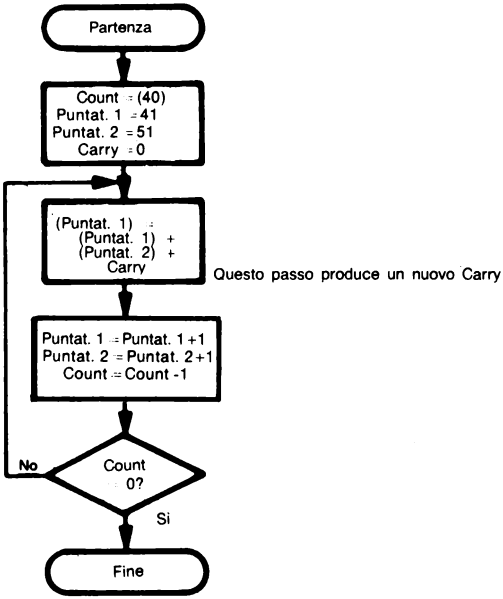
Problema Campione:

	(0040)	=	04
	(0041)	=	C3
	(0042)	=	A7
	(0043)	=	5B
	(0044)	=	2F
	(0051)	=	B8
	(0052)	=	35
	(0053)	=	DF
	(0054)	=	14
Risultato:	(0041)	=	7B
	(0042)	=	DD
	(0043)	=	3A
	(0044)	=	44

cioè

```
  2F5BA7C3
+ 14DF35B8
  --------
  443ADD7B
```

Diagramma di Flusso:



Programma Sorgente:

```
        LD      HL,40H   ;COUNT = LUNGHEZZA DI STRINGA (IN BYTE)
        LD      B,(HL)
        INC     HL       ;PUNTATORE 1 = PRIMA PAROLA DELLA STRINGA 1
        LD      DE,51H   ;PUNTATORE 2 = PRIMA PAROLA DELLA STRINGA 2
        AND     A        ;AZZERA IL CARRY PER PARTIRE
ADDW:   LD      A,(DE)   ;PRENDI LA PAROLA DALLA STRINGA 2
        ADC     A,(HL)   ;AGGIUNGI UNA PAROLA DALLA STRINGA 1
        LD      (HL),A   ;MEMORIZZA IL RISULTATO NELLA STRINGA 1
        INC     DE
        INC     HL
        DJNZ    ADDW
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)	
0000	21	LD	HL,40H
0001	40		
0002	00		
0003	46	LD	B,(HL)
0004	23	INC	HL
0005	11	LD	DE,51H
0006	51		
0007	00		
0008	A7	AND	A
0009	1A	ADDW: LD	A,(DE)
000A	8E	ADC	A,(HL)
000B	77	LD	(HL),A
000C	13	INC	DE
000D	23	INC	HL
000E	10	DJNZ	ADDW
000F	F9		
0010	76	HALT	

L'indirizzo relativo per DJNZ ADDW è:

$$\begin{array}{r} 09 \\ \underline{-10} \end{array} = \begin{array}{r} 09 \\ \underline{+F0} \\ F9 \end{array}$$

L'istruzione AND A viene utilizzata per azzerare il bit di Carry. Una qualsiasi altra operazione logica avrebbe lo stesso effetto. Il Carry deve essere azzerato, dal momento che non bisogna tener conto di alcun riporto nell'addizione dei byte meno significativi.

L'istruzione ADC, Somma col Carry, comprende il Carry delle precedenti parole nell'addizione. ADC è la sola istruzione nel ciclo che influenza il Carry. Ricordate che né INC né DJNZ eseguono ciò.

Sia il puntatore nella Coppia di Registri DE che quello in HL debbono essere aggiornati durante ogni ripetizione.

PRECISIONE DECIMALE IN BINARIO

Questa procedura può sommare numeri binari lunghi fino a 256 byte. Notate che i dieci bit binari corrispondono a tre cifre decimali, dato $2^{10} = 1024 \simeq 1000$. Così potete calcolare il numero di bit richiesti per fornire una certa precisione nelle cifre decimali. Per esempio, la precisione della cifra decimale dieci richiede:

$$(10 \text{ cifre}) \times \left(\frac{10 \text{ bit}}{3 \text{ cifre}}\right) = 33 \text{ bit}$$

ISTRUZIONI DI TRASFERIMENTO DI BLOCCO

Se stessimo solo trasferendo i dati da un punto della memoria ad un altro e senza neppure elaborarli, potremmo utilizzare la potente istruzione di trasferimento di blocco dello Z80, LDIR. Questa istruzione singola trasferisce un byte di dati dall'indirizzo contenuto in HL all'indirizzo individuato da DE, incrementa i puntatori in HL e DE, e decrementa il contatore di byte contenuto in BC. Essa ripete l'operazione di trasferimento finché BC raggiunge, decrementandosi, il valore zero. LDI è la stessa istruzione senza il fattore di ripetizione; LDD e LDDR sono trasferimenti ripetitivi e non ripetitivi rispettivamente che decrementano i puntatori invece di incrementarli.

Un programma per trasferire un numero di byte (LENGHT) da un punto della memoria (che inizia da PTR1) ad un altro (che inizia da PTR2) è riportato di seguito.

Trasferimento di Blocco

Scopo: Trasferire un blocco di dati lungo (BC) caratteri dall'indirizzo contenuto in HL all'indirizzo contenuto in DE.

Problema Campione:

```
          (HL) = 40
          (DE) = 50
          (BC) =  3
        (0040) = 31
        (0041) = 32
        (0042) = 33
        (0050) =  0
        (0051) =  0
        (0052) =  0
Risultato: (0050) = 31
        (0051) = 32
        (0052) = 33
```

Programma Sorgente:

```
LD      BC,LENGTH   ;COUNT = LUNGHEZZA
                    ;   DEL TRASFERIMENTO (IN BYTE)
LD      HL,PTR1     ;PUNTATORE 1 = INIZIO DELLA ZONA
                    ;   SORGENTE DI DATI
LD      DE,PTR2     ;PUNTATORE 2 = INIZIO DELLA ZONA
                    ;   DI DESTINAZIONE DEI DATI
LDIR
HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)	
0000	01	LD	BC,LENGTH
0001			
0002	LENGTH		
0003	21	LD	HL,PTR1
0004			
0005	PTR1		
0006	11	LD	DE,PTR2
0007			
0008	PTR2		
0009	ED	LDIR	
000A	B0		
000B	76	HALT	

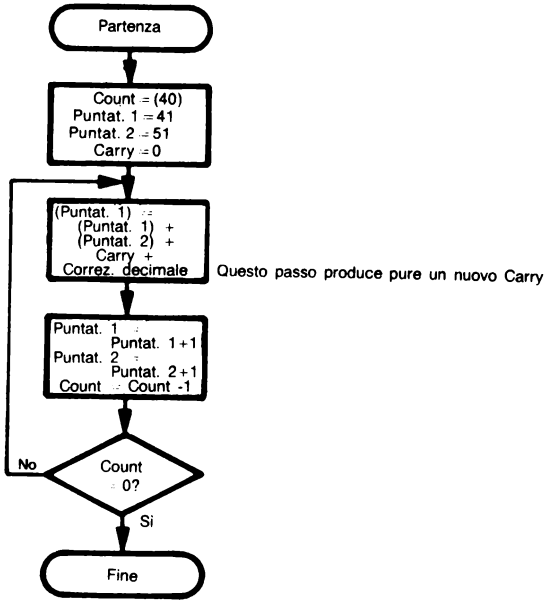
Provate ad implementare lo stesso programma senza l'istruzione LDIR. Quanti byte di memoria e cicli di clock sono necessari in ciascun caso?

Addizione Decimale

Scopo: Sommare due numeri decimali (BCD) multibyte. La lunghezza dei numeri è contenuta nella cella di memoria 0040, i numeri stessi iniziano (i bit meno significativi per primi) nelle celle 0041 e 0051, rispettivamente, e la somma sostituisce il numero che inizia dalla cella 0041.

Problema Campione:

```
            (0040) = 04
            (0041) = 85
            (0042) = 19
            (0043) = 70
            (0044) = 36
            (0051) = 59
            (0052) = 34
            (0053) = 66
            (0054) = 12
Risultato:  (0041) = 44
            (0042) = 54
            (0043) = 36
            (0044) = 49
cioè                 36701985
                    +12663459
                     49365444
```



Programma Sorgente:

```
        LD      HL,40H
        LD      B,(HL)      ;COUNT = LUNGHEZZA DELLE STRINGHE (IN BYTE)
        INC     HL          ;PUNTATORE 1 = PRIMA PAROLA DELLA STRINGA 1
        LD      DE,51H      ;PUNTATORE 2 = PRIMA PAROLA DELLA STRINGA 2
        AND     A           ;AZZERA IL CARRY PER PARTIRE
DECAD:  LD      A,(DE)      ;PRENDI I DIGIT 2 DECIMALE DALLA STRINGA 2
        ADC     A,(HL)      ;AGGIUNGI UNA COPPIA DI DIGIT DALLA STRINGA 1
        DAA                 ;FAI L'ADDIZIONE DECIMALE
        LD      (HL),A      ;MEMORIZZA IL RISULTATO NELLA STRINGA 1
        INC     DE
        INC     HL
        DJNZ    DECAD
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)		
0000	21		LD	HL,40H
0001	40			
0002	00			
0003	46		LD	B,(HL)
0004	23		INC	HL
0005	11		LD	DE,51H
0006	51			
0007	00			
0008	A7		AND	A
0009	1A	DECAD:	LD	A,(DE)
000A	8E		ADC	A,(HL)
000B	27		DAA	
000C	77		LD	(HL),A
000D	13		INC	DE
000E	23		INC	HL
000F	10		DJNZ	DECAD
0010	F8			
0011	76		HALT	

ADATTAMENTO DECIMALE

L'istruzione di Adattamento Decimale (DAA) utilizza il bit di Carry (C) e di Half Carry (H) per correggere le seguenti situazioni:

1) La somma di due cifre è compresa tra 10 e 15 incluso. In questo caso, si deve aggiungere 6 alla somma per fornire il risultato esatto, cioè:

```
     0101     (5)
   + 1000     (8)
     1101     (D)
   + 0110
0001 0011     (13 in BCD, che è corretto)
```

2) La somma di due cifre è maggiore o uguale a 16. In questo caso il risultato è un numero esatto in BCD ma minore di sei rispetto a quanto dovrebbe essere, cioè:

```
     1000     (8)
   + 1001     (9)
0001 0001     (11 in BCD)
   + 0110
0001 0111     (17 in BCD, che è corretto)
```

In ambedue i casi bisogna sommare sei. Tuttavia, il caso 1 può essere riconosciuto dal fatto che la somma non è una cifra BCD; è compresa tra 10 e 15 (A ed F esadecimale). Il caso 2 può essere riconosciuto solamente dal fatto che il Carry (cifra più significativa) o Half-Carry (cifra meno significativa) è stato portato ad 1, dato che il risultato è un numero valido BCD. DAA è la sola istruzione che utilizza il bit di Half-Carry. Notate che DAA agisce solamente sull'Accumulatore.

FLAG DI ADD/SUBTRACT

Il microprocessore Z80 ha anche un flag che fa la distinzione tra le istruzioni di Addizione (ADD, ADC) e quelle di Sottrazione (SUB, SUBC). Questo flag, chiamato flag di Addizione/Sottrazione o flag N, viene azzerato da tutte le istruzioni di Addizione e posizionato ad 1 da tutte le istruzioni di Sot-

trazione. Il solo uso di questo flag è quello di permettere alla istruzione DAA di convertire in modo corretto l'addizione binaria in addizione BCD e la sottrazione binaria in sottrazione BCD. I microprocessori 8080 e 8085 non posseggono un flag N, e pertanto le loro istruzioni DAA agiscono propriamente solo dopo l'addizione.

DAA può essere usata solo dopo istruzioni che pongono il loro risultato nell'Accumulatore e che influenzano correttamente i flag di Carry, Half-Carry e Addizione/Sottrazione. Così, non si può usare DAA dopo INC (dato che INC non influenza il Carry), DEC, o qualsiasi altra istruzione a due byte che pone il suo risultato nei registri indice o nella Coppia di Registri HL.

PRECISIONE IN BINARIO ED IN BCD

Questa procedura è in grado di sommare numeri decimali (BCD) di qualsiasi lunghezza. In questo caso sono necessari quattro bit binari per ogni cifra decimale, cosicché la precisione a dieci cifre richiede:

$$10 \times 4 = 40 \text{ bit}$$

rispetto ai 33 bit del caso binario. Ciò coincide essenzialmente con l'uso di cinque byte ad 8 bit invece di quattro. La procedura decimale richiede anche un tempo maggiore per byte a causa dell'istruzione supplementare DAA.

Moltiplicazione Binaria ad 8 bit

Scopo: Moltiplicare il numero ad 8 bit privo di segno contenuto nella locazione di memoria 0040 con il numero di bit bit privo di segno contenuto nella locazione 0041. Porre gli otto bit meno significativi del risultato nella cella di memoria 0042 e gli otto bit più significativi nella cella 0043.

Problemi Campione:

a.
(0040) = 03
(0041) = 05
Risultato: (0042) = 0F
(0043) = 00
oppure in decimale 3 x 5 = 15

b.
(0040) = 6F
(0041) = 61
Risultato: (0042) = 0F
(0043) = 2A
oppure 111 x 97 = 10.767

Potete eseguire una moltiplicazione con un computer nell'identico modo con cui eseguite a mano una lunga moltiplicazione. Dato che i numeri sono binari, l'unico problema è quello di moltiplicare per 0 o per 1; la moltiplicazione per zero dà ovviamente zero come risultato, mentre la moltiplicazione per uno produce lo stesso numero di partenza (il moltiplicando). Così ogni passo in una moltiplicazione binaria può essere ridotto alla seguente operazione.

ALGORITMO DI MOLTIPLICAZIONE

Se il bit corrente nel moltiplicatore è 1, sommare il moltiplicando al prodotto parziale.

L'unico problema restante è quello di assicurare gli incolonnamenti ogni volta in un modo corretto. Le operazioni seguenti realizzano questo compito.

1) Spostare il moltiplicatore di un bit verso sinistra in modo tale che il bit da esaminare venga posto nel Carry.
2) Spostare il prodotto di un bit verso sinistra in modo che la successiva addizione sia incolonnata in modo esatto.

Il processo completo per la moltiplicazione binaria è il seguente:

1° Passo - Inizializzazione
Prodotto = 8
Contatore = 8

2° Passo - Spostamento del Prodotto in modo tale da incolonnare in modo esatto
Prodotto = 2 X Prodotto (LSB = 0)

3° Passo - Spostamento del Moltiplicatore in modo che il bit vada a finire nel Carry
Moltiplicatore = 2 X Moltiplicatore

4° Passo - Somma del Moltiplicando col Prodotto se il Carry è 1
Se il Carry = 1, Prodotto = Prodotto + Moltiplicando

5° Passo - Decremento del contatore e controllo dello zero
Contatore = - 1
Se il Contatore $\neq$ 0 ripresa dal 2° Passo

Nel caso del Problema Campione b, dove il moltiplicatore è 61 (esadecimale) e il moltiplicando è 6F (esadecimale) il processo funziona come segue:

Inizializzazione

Prodotto	0000
Moltiplicatore	61
Moltiplicando	6F
Contatore	08

Dopo la prima ripetizione dei passi 2 - 5:

Prodotto	0000
Moltiplicatore	C2
Moltiplicando	6F
Contatore	07
Carry del Moltiplicatore	0

Dopo la seconda ripetizione:

Prodotto	006F
Moltiplicatore	84
Moltiplicando	6F
Contatore	05
Carry dal Moltiplicatore	1

Dopo la terza ripetizione:

Prodotto	014D
Moltiplicatore	08
Moltiplicando	6F
Contatore	05
Carry dal Moltiplicatore	1

Dopo la quarta ripetizione:

Prodotto	029A
Moltiplicatore	10
Moltiplicando	6F
Contatore	04
Carry dal moltiplicatore	0

Dopo la quinta ripetizione:

Prodotto	0534
Moltiplicatore	20
Moltiplicando	6F
Contatore	03
Carry dal Moltiplicatore	0

Dopo la sesta ripetizione:

Prodotto	0A68
Moltiplicatore	40
Moltiplicando	6F
Contatore	02
Carry dal moltiplicatore	0

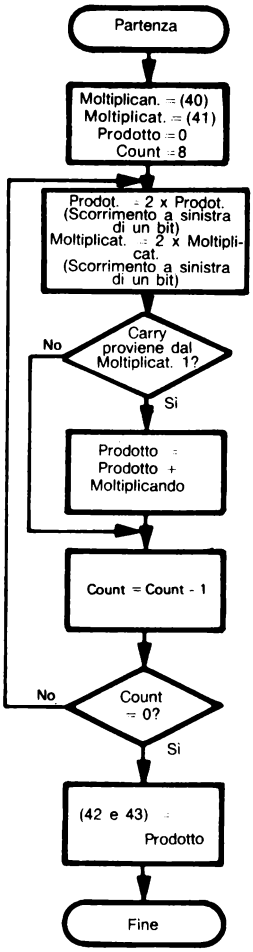
Dopo la settima ripetizione:

Prodotto	14D0
Moltiplicatore	80
Moltiplicando	6F
Contatore	01
Carry dal Moltiplicatore	0

Dopo l'ottava ripetizione:

Prodotto	2A0F
Moltiplicatore	00
Moltiplicando	6F
Contatore	00
Carry dal Moltiplicatore	1

Diagramma di Flusso:



Programma Sorgente:

```
        LD      HL,40H
        LD      E,(HL)          ;PRENDI IL MOLTIPLICANDO
        LD      D,0             ;ESTENSIONE A 16 BIT
        INC     HL
        LD      A,(HL)          ;PRENDI IL MOLTIPLICATORE
        LD      HL,0            ;PRODOTTO = ZERO
        LD      B,8             ;COUNT = LUNGHEZZA DEI BIT DEL MOLTIPLICATORE
MULT:   ADD     HL,HL           ;SCORRIMENTO DEL PRODOTTO A SINISTRA DI UN BIT
        RLA                     ;SCORRIMENTO DEL MOLTIPLICATORE
                                ;   A SINISTRA DI UN BIT
        JR      NC,CHCNT        ;IL CARRY PROVIENE DAL MOLTIPLICATORE?
        ADD     HL,DE           ;SÌ, SOMMA IL MOLTIPLICANDO AL PRODOTTO
CHCNT:  DJNZ    MULT
        LD      (42H),HL        ;SALVA IL PRODOTTO IN MEMORIA
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)		
0000	21		LD	HL,40H
0001	40			
0002	00			
0003	5E		LD	E,(HL)
0004	16		LD	D,0
0005	00			
0006	23		INC	HL
0007	7E		LD	A,(HL)
0008	21		LD	HL,0
0009	00			
000A	00			
000B	06		LD	B,8
000C	08			
000D	29	MULT:	ADD	HL,HL
000E	17		RLA	
000F	30		JR	NC,CHCNT
0010	01			
0011	19		ADD	HL,DE
0012	10	CHCNT:	DJNZ	MULT
0013	F9			
0014	22		LD	(42H),HL
0015	42			
0016	00			
0017	76		HALT	

Notate che il moltiplicando deve essere ampliato a 16 bit mediante l'azzeramento del Registro D in modo che esso possa essere sommato al prodotto utilizzando l'istruzione ADD HL, DE.

L'istruzione ADD HL, HL si comporta come uno spostamento logico verso sinistra a 16 bit per il prodotto a 16 bit.

In questo programma, le istruzioni dello Z80 che operano su 16 bit trattano dei dati piuttosto che degli indirizzi. LD HL,0 viene usata per inizializzare il prodotto; ADD HL, HL per eseguire uno spostamento logico verso sinistra a 16 bit; ADD HL, DE per sommare il moltiplicando al prodotto parziale; e LD (42H), HL per immagazzinare il risultato in memoria. Bisogna badare ad ampliare le quantità di 8 bit (come il moltiplicando in questo esempio) a quantità di 16 bit. Notate che non è possibile usare le facilitazioni a 16 bit contemporaneamente per gli indirizzamenti e la manipolazione di dati. Tuttavia, se non v'è altra necessità per l'uso dei registri alternativi, si potrebbero salvare in questi ultimi i vecchi contenuti dei registri normali di lavoro e ripristinarli in seguito mediante l'istruzione EXX. Questa istruzione scambia il contenuto delle Coppie di Registri BC, DE e HL con i contenuti dei loro equivalenti alternativi in un tempo corrispondente a quattro cicli di clock.

Oltre al suo ovvio utilizzo nei calcolatori e nei terminali per punti di vendita, la moltiplicazione è una parte chiave di quasi tutte le procedure di segnale e gli algoritmi di controllo. La velocità con la quale le moltiplicazioni possono essere eseguite determina l'utilità della CPU nel controllo di processo, nella rivelazione di segnale e nell'analisi di segnale.

L'algoritmo necessita di 390÷440 cicli di clock per eseguire la moltiplicazione su un microprocessore Z80. Il tempo preciso dipende dal numero di bit pari ad uno nel moltiplicatore. Altri algoritmi possono essere in grado di ridurre il tempo medio di esecuzione, ma 400 cicli di clock sarà un tempo di esecuzione tipico per una moltiplicazione in software[2].

Divisione Binaria ad 8 Bit

Scopo: Dividere il numero privo di segno a 16 bit contenuto nelle celle di memoria 0040 e 0441 (i bit più significativi in 0041) per il numero privo di segno ad 8 bit contenuto nella cella di memoria 0042. I numeri sono normalizzati in modo che 1) i bit più significativi sia del dividendo che del divisore siano zero e 2) il numero presente nella cella 0042 sia maggiore del numero presente nella cella 0041, vale a dire il quoziente sia un numero ad 8 bit. Memorizzare il quoziente nella cella 0043 e il resto della divisione in 0044.

Problemi Campione:

a.
(0040) = 40 (64 decimale)
(0041) = 00
(0042) = 08
Risultato: = (0043) = 08
(0044) = 00
cioè 64/8 = 8

b.
(0040) = 6D (12.909 in decimale)
(0041) = 32
(0042) = 47 (71 decimale)
Risultato: = (0043) = B5 (181 decimale)
(0044) = 3A (58 decimale)
cioè 12.909/71 = 181 con resto di 58

ALGORITMO DELLA DIVISIONE

Si può eseguire la divisione col computer proprio come la si eseguisse con carta e matita, cioè utilizzando le sottrazioni di prova. Dato che i numeri sono binari, l'unico problema è se il bit nel quoziente è 0 oppure 1, cioè se il divisore può essere sottratto oppure no da quanto rimane dal dividendo. Ogni passo in una divisione binaria può essere ridotto alla seguente operazione:

> Se il divisore può essere sottratto dagli otto bit più significativi del dividendo senza un prèstito, il bit corrispondente nel quoziente è pari a 1; altrimenti esso è 0.

Il solo problema restante è quello di incolonnare appropriatamente il dividendo e il quoziente. Questo può essere realizzato spostando logicamente di un bit verso sinistra il dividendo ed il quoziente prima di ciascuna sottrazione di prova. Il dividendo e il quoziente possono condividere un registro a 16 bit, dato che la procedura azzera un bit del dividendo nello stesso istante in cui esso determina un bit del quoziente.

Il processo completo per la divisione binaria è:

1° Passo - Inizializzazione
Quoziente = 0
Contatore = 8

2° Passo - Spostamento del Dividendo e del Quoziente in modo tale da incolonnare in modo esatto:
Dividendo = 2 X Quoziente
Quoziente = 2 X Quoziente

3° Passo - Esecuzione della Sottrazione di prova. Se non c'è Prestito sommare 1 al Quoziente:
Se gli 8 MSB del Dividendo $\geq$ Divisore allora gli MSB del Dividendo = gli MSB del Dividendo - Divisore
Quoziente = Quoziente + 1.

4° Passo - Decremento del contatore e controllo dello zero:
Contatore = Contatore - 1
se il Contatore $\neq$ 0, ripresa dal 2° Passo
Resto = gli 8 MSB del Dividendo

Nel caso del problema campione b, dove il dividendo è 326D (esadecimale) e il divisore è 47 (esadecimale), il processo funziona come segue:

Inizializzazione:

Dividendo	326D
Divisore	47
Quoziente	00
Contatore	00

Dopo la prima ripetizione dei Passi 2 - 4:
(Notate che il dividendo viene spostato prima della sottrazione di prova)

Dividendo	1DDA
Divisore	47
Quoziente	01
Contatore	07

Dopo la seconda ripetizione dei Passi 2 - 4:

Dividendo	3BB4
Divisore	47
Quoziente	02
Contatore	06

Dopo la terza ripetizione:

Dividendo	3068
Divisore	47
Quoziente	05
Contatore	05

Dopo la quarta ripetizione:

Dividendo	19D0
Divisore	47
Quoziente	0B
Contatore	04

Dopo la quinta ripetizione:

Dividendo	33A0
Divisore	47
Quoziente	16
Contatore	03

Dopo la sesta ripetizione:

Dividendo	2040
Divisore	47
Quoziente	2D
Contatore	02

Dopo la settima ripetizione:

Dividendo	4080
Divisore	47
Quoziente	5A
Contatore	01

Dopo la ottava ripetizione:

Dividendo	3A00
Divisore	47
Quoziente	B5
Contatore	00

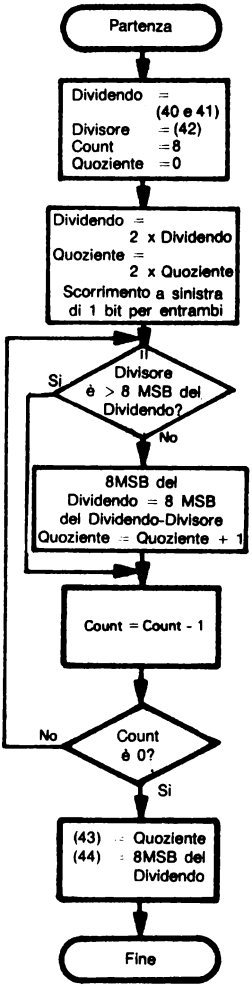
Così il quoziente è B5 e il resto è 3A.

Gli MSB del dividendo e del divisore si suppongono uguali a zero in modo da semplificare i calcoli (lo spostamento precedente la sottrazione di prova porterebbe altrimenti l'MSB del dividendo nel Carry). I problemi che non sono posti in questa forma debbono essere semplificati separando le parti del quoziente che provocherebbero un «overflow» su una parola ad 8 bit. Per esempio:

$$\frac{1024}{3} = \frac{400 \text{ (esadecimale)}}{3} = 100 + \frac{100 \text{ (esadecimale)}}{3}$$

Questo problema è ora nella forma appropriata. Può essere necessaria una divisione supplementare.

Diagramma di Flusso:



Programma Sorgente:

```
        LD      HL,(40H)
        LD      A,(42H)
        LD      C,A
        LD      B,8
DIV:    ADD     HL,HL
        LD      A,H
        SUB     C
        JR      C,CNT
        LD      H,A
        INC     L
CNT:    DJNZ    DIV
        LD      (43H),HL
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)		
0000	2A		LD	HL,(40H)
0001	40			
0002	00			
0003	3A		LD	A,(42H)
0004	42			
0005	00			
0006	4F		LD	C,A
0007	06		LD	B,8
0008	08			
0009	29	DIV:	ADD	HL,HL
000A	7C		LD	A,H
000B	91		SUB	C
000C	38		JR	C,CNT
000D	02			
000E	67		LD	H,A
000F	2C		INC	L
0010	10	CNT:	DJNZ	DIV
0011	F7			
0012	22		LD	(43H),HL
0013	43			
0014	00			
0015	76		HALT	

La Coppia di Registri HL contiene sia il dividendo che il quoziente. Il quoziente sostituisce semplicemente il dividendo nel Registro L poiché il dividendo viene spostato logicamente verso sinistra.

Per problemi di divisioni più lunghe, si potrebbe usare l'istruzione SBC H, che sottrae il contenuto di una coppia di registri ed il contenuto del Carry dal contenuto della Coppia di Registri HL.

L'istruzione INC L posiziona il bit meno significativo del quoziente ad 1, dato che ADD HL, HL ha precedentemente azzerato quel bit.

La divisione è necessaria nei calcolatori, nei terminali, nel controllo d'errore nelle comunicazioni, negli algoritmi di controllo, e molte altre applicazioni.

Questo algoritmo necessita di 400÷430 cicli di clock per eseguire la divisione su un microprocessore Z80. Il tempo preciso dipende dal numero di bit uguali ad uno nel quoziente. Altri algoritmi possono ridurre il tempo medio di esecuzione, ma 400 cicli di clock sarà ancora un valore tipico per una divisione in software. Alcuni riferimenti bibliografici riportati al termine di questo capitolo trattano di metodi più veloci per l'implementazione della divisione.

Numeri di Autocontrollo
Double Add Double Mod 10

Scopo: Calcolare una cifra di controllo di somma (checksum) da una stringa di cifre BCD. La lunghezza della stringa di cifre (numero di parole) è contenuta nella cella di memoria 0041; la stringa di cifre (2 cifre BCD a parola) inizia nella cella di memoria 0042. Calcolare la cifra di controllo di somma con la tecnica di «Double Add Double Mod 10»[3] e memorizzarla nella cella 0040.

NUMERI CHE SI AUTOCONTROLLANO

La tecnica di «Double Add Double Mod 10» funziona come segue:

1) Azzerare il controllo di somma per cominciare.
2) Moltiplicare la prima cifra per due e sommare il risultato al controllo di somma.
3) Sommare la cifra successiva al controllo di somma.
4) Continuare il processo alternato finché non si sono usate tutte le cifre.
5) La cifra meno significativa del controllo di somma è la cifra di autocontrollo.

Le cifre di autocontrollo vengono solitamente sommate ai numeri di identificazione sulle carte di credito, sui cartellini d'inventario, sui bagagli, sui pacchi, ecc., quando esse sono trattate da sistemi computerizzati. Esse possono anche essere usate nei messaggi guida, file di identificazione, e altre applicazioni. Lo scopo delle cifre è quello di minimizzare gli errori all'ingresso quali cifre invertite (69 invece di 96), cifre che si spostano (7260 invece di 3726) cifre mancanti di un'unità (65 invece di 54), ecc. Si può verificare automaticamente il numero di autocontrollo per correttezza sull'ingresso e può eliminare immediatamente parecchi errori.

L'analisi dei metodi di autocontrollo è piuttosto complessa. Per esempio, un semplice controllo di somma non metterà in evidenza errori di inversione ($4 + 9 = 9 + 4$). L'algoritmo «Double Add Double» scoprirà i semplici errori d'inversione ($2 \times 4 + 9 = 17 \neq 2 \times 9 + 4$); ma perderà di vista alcuni errori, quali inversioni su numeri dispari di cifre (367 invece di 763). Tuttavia, questo metodo scoprirà molti errori comuni! La validità di un metodo dipende da quali errori rivelerà e dalla probabilità di errori particolari in una applicazione.

Per esempio, se la stringa di cifre è

549321

il risultato sarà:

Controllo di somma $= 5 \times 2 + 4 + 9 \times 2 + 3 + 2 \times 2 + 1 = 40$
Cifra di autocontrollo = 0 (cifra meno significativa di un controllo di somma)

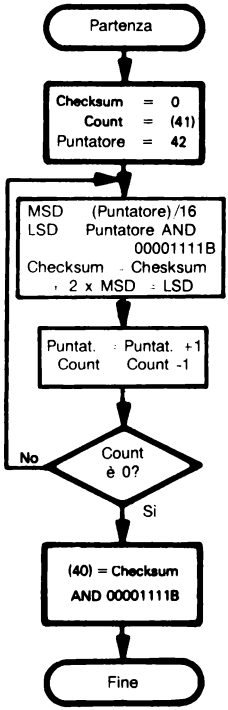
Notate che un ingresso errato come 543921 produrrebbe una cifra diversa di autocontrollo (4) ma ingressi errati come 049321 o 945321 non sarebbero rivelati.

Problemi Campione:

a. (0041) = 03
(0042) = 36
(0043) = 68
(0044) = 51
Risultato: Checksum = 3 x 2 + 6 + 6 x 2 + 8 + 5 x 2 + 1 = 43
(0040) = 03

b. (0041) = 04
(0042) = 50
(0043) = 29
(0044) = 16
(0045) = 83
Risultato: Checksum = 5 x 2 + 0 + 2 x 2 + 9 + 1 x 2 + 6 + 8 x 2 + 3 = 50
(0040) = 00

Diagramma di Flusso:



Programma Sorgente:

```
        LD      A,(41H)     ;COUNT = LUNGHEZZA DELLA STRINGA IN BYTE
        LD      B,A
        LD      C,0         ;CHECKSUM = 0
        LD      HL,42H      ;PUNTA ALL'INIZIO DELLA STRINGA DI CIFRE
CHDIG:  LD      A,(HL)      ;ACCETTA DUE CIFRE BCD DALLA STRINGA
        LD      D,A         ;SALVA UNA COPIA
        RRA                 ;RICAVA MSD MEDIANTE SCORRIMENTO
                            ;   E MASCHERATURA
        RRA
        RRA
        RRA
        AND     00001111B
        ADD     A,A         ;MOLTIPLICA PER DUE MSD
        DAA                 ;RENDI DECIMALE IL DOPPIO DI MSD
        ADD     A,C         ;SOMMA MSD RADDOPPIATO A CHECKSUM
        DAA                 ;RENDI DECIMALE CHECKSUM
        LD      C,A
        LD      A,D         ;ACCETTA LA CIFRA MENO SIGNIFICATIVA
        AND     00001111B   ;(MASCHERA MSD)
        ADD     A,C         ;SOMMA LSD A CHECKSUM
        DAA                 ;RENDI DECIMALE CHECKSUM
        LD      C,A
        INC     HL
        DJNZ    CHDIG
        AND     00001111B   ;MASCHERA LA CIFRA DI AUTOCONTROLLO
        LD      (40H),A     ;SALVA LA CIFRA DI AUTOCONTROLLO
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)		
0000	3A		LD	A,(41H)
0001	41			
0002	00			
0003	47		LD	B,A
0004	0E		LD	C,0
0005	00			
0006	21		LD	HL,42H
0007	42			
0008	00			
0009	7E	CHDIG:	LD	A,(HL)
000A	57		LD	D,A
000B	1F		RRA	
000C	1F		RRA	
000D	1F		RRA	
000E	1F		RRA	
000F	E6		AND	00001111B
0010	0F			
0011	87		ADD	A,A
0012	27		DAA	
0013	81		ADD	A,C
0014	27		DAA	
0015	4F		LD	C,A
0016	7A		LD	A,D
0017	E6		AND	00001111B
0018	0F			
0019	81		ADD	A,C
001A	27		DAA	
001B	4F		LD	C,A
001C	23		INC	HL
001D	10		DJNZ	CHDIG
001E	EA			
001F	E6		AND	00001111B
0020	0F			
0021	32		LD	(40H),A
0022	40			
0023	00			
0024	76		HALT	

Le cifre vengono trasferite mediante lo spostamento e la mascheratura. Sono necessari quattro spostamenti verso destra per estrarre la cifra più significativa.

Un adattamento decimale (DAA) deve seguire un'addizione per produrre l'esatto risultato decimale. Una sola DAA dopo una serie di addizioni non basterà (provate!). Ricordate che DAA lavora solamente con l'Accumulatore.

Non ci sono problemi con i riporti da una somma decimale, dato che la procedura utilizza in ogni caso solo la cifra meno significativa del controllo di somma.

ISTRUZIONI DI SCORRIMENTO DECIMALE

Un metodo alternativo (e superiore) è quello di usare l'istruzione di spostamento decimale RLD dello Z80. Questa istruzione è uno spostamento a 4 bit che trasferisce il contenuto dei quattro bit meno significativi della cella di memoria indirizzata da HL nei quattro bit più significativi di quella stessa cella, il contenuto precedente dei quattro bit più significativi della stessa nei quattro bit meno significativi dell'Accumulatore, e il contenuto precedente dei quattro bit meno significativi dell'Accumulatore nei quattro bit meno significativi della cella di memoria. Pertanto, RLD non solo trasferisce una singola cifra nell'Accumulatore, ma sposta anche la cifra successiva in modo tale che essa possa essere trasferita nell'Accumulatore con la RLD seguente. La Figura 8-1 mostra un esempio di come funzioni RLD; RRD è la stessa istruzione tranne che lo spostamento viene effettuato verso destra anziché verso sinistra.

L'algoritmo «Double Add Mod 10» può essere implementato come segue utilizzando RLD:

Programma Sorgente:

```
        LD      A,(41H)     ;COUNT = LUNGHEZZA DELLE STRINGHE (IN BYTE)
        LD      B,A
        LD      C,0         ;CHECKSUM = 0
        LD      HL,42H      ;PUNTA ALL'INIZIO DELLA STRINGA DI CIFRE
CHDIG:  SUB     A           ;AZZERA MSD
        RLD                 ;ACCETTA MSD DALLA STRINGA
        ADD     A,A         ;RADDOPPIA MSD
        DAA                 ;RENDI DECIMALE MSD RADDOPPIATO
        ADD     A,C         ;SOMMA MSD RADDOPPIATO A CHECKSUM
        DAA                 ;RENDI DECIMALE CHECKSUM
        LD      C,A
        SUB     A           ;AZZERA MSD
        RLD                 ;ACCETTA LSD DALLA STRINGA
        ADD     A,C         ;SOMMA LSD A CHECKSUM
        DAA                 ;RENDI DECIMALE CHECKSUM
        LD      C,A
        INC     HL
        DJNZ    CHDIG
        AND     00001111B   ;MASCHERA LA CIFRA DI AUTOCONTROLLO
        LD      (40H),A     ;SALVA LA CIFRA DI AUTOCONTROLLO
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)		
0000	3A		LD	A,(41H)
0001	41			
0002	00			
0003	47		LD	B,A
0004	0E		LD	C,0
0005	00			
0006	21		LD	HL,42H
0007	42			
0008	00			
0009	97	CHDIG:	SUB	A
000A	ED		RLD	
000B	6F			
000C	87		ADD	A,A
000D	27		DAA	
000E	81		ADD	A,C
000F	27		DAA	
0010	4F		LD	C,A
0011	97		SUB	A
0012	ED		RLD	
0013	6F			
0014	81		ADD	A,C
0015	27		DAA	
0016	4F		LD	C,A
0017	23		INC	HL
0018	10		DJNZ	CHDIG
0019	EF			
001A	E6		AND	00001111B
001B	0F			
001C	32		LD	(40H),A
001D	40			
001E	00			
001F	76		HALT	

Si potrebbe perfezionare ancora di più questo programma (è già più corto della versione precedente). Dato che si fa cadere in ogni caso la cifra più significativa alla fine, non c'è alcuna ragione per azzerarla ogni volta con l'istruzione SUB A.

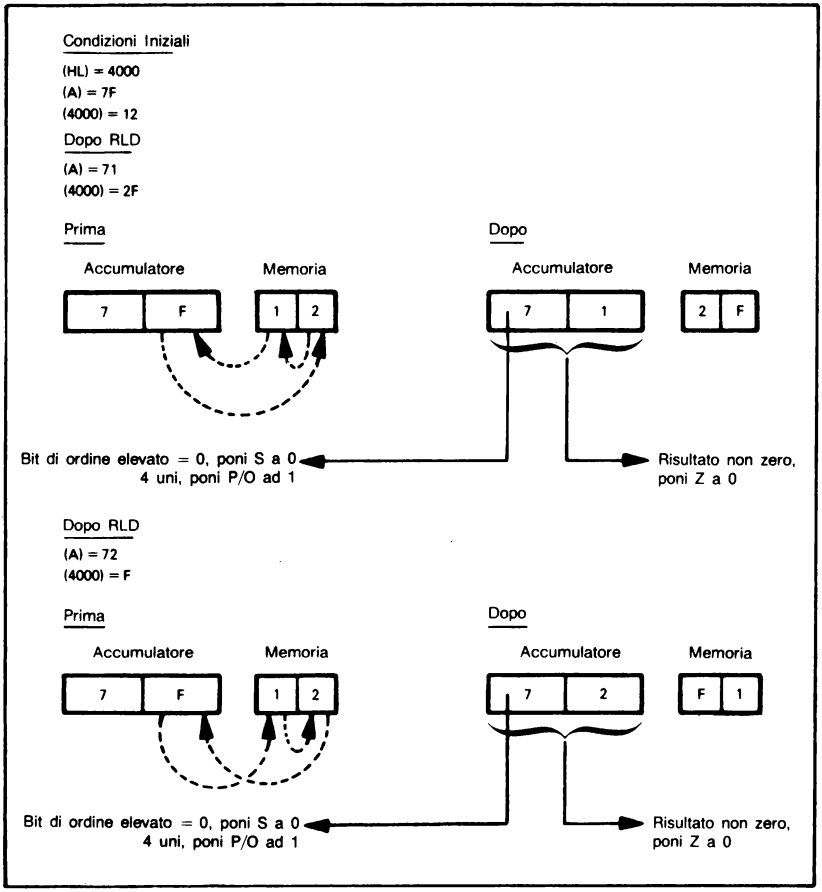


Figura 8-1. Esempi degli Spostamenti di Cifra dello Z80

RADDOPPIO E DIMEZZAMENTO DI NUMERI BINARI

Si può raddoppiare un numero decimale (nell'Accumulatore) sommandolo a se stesso ed eseguendo poi una correzione decimale, cioè:

```
ADD     A       ;RADDOPPIA IL NUMERO
DAA             ;E RENDI DECIMALE IL RISULTATO
```

Ricordate che l'Accumulatore può contenere solamente cifre decimali valide nel campo di variazione 0-99.

Non si può usare SLA A (Spostamento Aritmetico verso Sinistra di A) poiché quell'istruzione azzera sempre il flag Half-Carry (solo le istruzioni di Addizione e di Sottrazione posizionano H in modo corretto).

Si può dividere un numero decimale per due semplicemente spostandolo logicamente verso destra e sottraendo quindi tre da qualsiasi cifra uguale o maggiore di otto (dato che 10 in BCD vale 16 in binario). Il seguente programma divide un numero decimale contenuto nella cella di memoria 0040 per due e pone il risultato nella cella 0041.

```
        LD      A,(40H)     ;ACCETTA NUMERO DECIMALE
        SRL     A           ;DIVIDI PER 2 IN BINARIO
        BIT     3,A         ;LA CIFRA MENO SIGNIFICATIVA È MAGGIORE
                            ;  O UGUALE AD 8?
        JR      Z,DONE
        SUB     3           ;SÌ, SOTTRAI 3 PER CORREZIONE DECIMALE
DONE:   LD      (41H),A     ;MEMORIZZA IL NUMERO DIVISO PER 2
        HALT
```

Provate questo programma ed il metodo sui numeri decimali 28, 30 e 37. Comprendete perchè funziona?

ARROTONDAMENTO BINARIO

L'arrotondamento è semplice e i numeri sono binari o decimali. Un numero binario può essere arrotondato come segue:

> Se il bit più significativo da eliminare è pari ad 1, sommare 1 ai bit rimanenti. Altrimenti, lasciare stare i bit rimanenti.

Questa regola funziona poiché 1 è a mezza strada tra 0 e 10 in binario; come 5 è una via di mezzo in decimale (notate che 0,5 decimale = 0,1 binario).

Così, il seguente programma arrotonderà un numero a 16 bit contenuto nelle locazioni di memoria 0040 e 0041 (i MSB in 0041) a un numero ad 8 bit da porre nella cella 0041.

```
        LD      HL,40H
        BIT     7,(HL)      ;L'MSB DEL BYTE EXTRA È 1?
        JR      Z,DONE
        INC     HL          ;NO, ARROTONDA
        INC     (HL)
DONE:   HALT
```

Se il numero è più lungo di 16 bit, l'arrotondamento deve propagarsi necessariamente agli altri byte.

ARROTONDAMENTO DECIMALE

L'arrotondamento decimale è leggermente più difficoltoso dato che il punto di incrocio è ora BCD 50 e l'arrotondamento deve produrre un risultato decimale. La regola è:

> Se la cifra più significativa da eliminare è pari a 5 o più, somma 1 alle cifre rimanenti.

Il programma seguente arrotonderà un numero di 4 cifre BCD contenuto nelle celle di memoria 0040 e 0041 (MSB in 0041) ad un numero di 2 cifre BCD nella cella di memoria 0041.

```
        LD      HL,40H
        LD      A,(HL)      ;IL BYTE DA TRASCURARE È 50 O MAGGIORE?
        CP      50H
        JR      C,DONE
        INC     HL          ;SÌ, ARROTONDA MSB
        LD      A,(HL)
        ADD     A,1
        DAA                 ;RENDI DECIMALI LE CIFRE
        LD      (HL),A
DONE:   HALT
```

Ricordate che l'istruzione DAA lavora solo su numeri contenuti nell'Accumulatore. In questo caso, si potrebbe arrotondare mediante l'istruzione INC A, dato che si sa che il Carry è zero

(perchè? — ricordate l'istruzione JR). Normalmente, è necessaria la sequenza ADD A, 1 seguita da DAA, dato che INC A non influenza il flag di Carry.

PROPAGAZIONE DEL SEGNO

Molto spesso quando si eseguono operazioni aritmetiche con segno in complemento a due su parole multibyte, è necessario propagare il bit di segno attraverso i byte di ordine elevato. Questa operazione può essere eseguita in modo diretto se, com'è solitamente, il segno è contenuto nel Carry. L'istruzione SBC A,A, ha come effetto la propagazione dello stato del Carry dall'inizio alla fine di una parola. Dato che A-A è sempre uguale a 0, SBC A,A equivale alla sottrazione del Carry da 0 e può dare solamente i valori 0 ed FFH.

PROBLEMI

1) Sottrazione In Multi-precisione

Scopo: Sottrarre un numero multi-byte da un altro. La lunghezza dei numeri è contenuta nella cella di memoria 0040, gli stessi numeri iniziano (i bit meno significativi per primi) nelle celle di memoria 0041 e 0051 rispettivamente e la differenza sostituisce il numero che inizia nella cella 0041. Sottrarre il numero che inizia nella cella 0051 da quello che inizia nella cella 0041.

Problema Campione:

```
            (0040) = 04
            (0041) = C3
            (0042) = A7
            (0043) = 5B
            (0044) = 2F
            (0051) = B8
            (0052) = 35
            (0053) = DF
            (0054) = 14
Risultato:  (0041) = 0B
            (0042) = 72
            (0043) = 7C
            (0044) = 1A
cioè                  2F5BA7C3
                     +14DF35B8
                     ---------
                      1A7C720B
```

2) Sottrazione Decimale

Scopo: Sottrarre un numero decimale (BCD) multibyte da un altro. La lunghezza dei numeri è contenuta nella cella di memoria 0040, i numeri stessi iniziano (i bit meno significativi per primi) nelle celle 0041 e 0051, rispettivamente, e la differenza sostituisce il numero che inizia nella cella 0041. Sottrarre il numero che inizia in 0051 da quello che inizia in 0041.

Problema Campione:

```
            (0040) = 04
            (0041) = 85
            (0042) = 19
            (0043) = 70
            (0044) = 36
            (0051) = 59
            (0052) = 34
            (0053) = 66
            (0054) = 12
Risultato:  (0041) = 26
            (0042) = 85
            (0043) = 03
            (0044) = 24
cioè                 36701985
                   - 12663459
                     --------
                     24038526
```

3) Moltiplicazione Binaria di 16 Bit per 8 Bit

Scopo: Moltiplicare il numero a 16 bit privo di segno contenuto nella locazione di memoria 0040 e 0041 (i MSB in 0041) per il numero ad 8 bit privo di segno contenuto nella cella 0042. Memorizzare il risultato nelle locazioni da 0043 a 0045, con i bit più significativi nella locazione 0045.

Problemi Campione:

```
a.          (0040) = 03
            (0041) = 00
            (0042) = 05
Risultato:  (0043) = 0F
            (0044) = 00
            (0045) = 00
cioè                 3 x 5 = 15
b.          (0040) = 6F
            (0041) = 72   (29.295 decimale)
            (0042) = 61   (97 decimale)
Risultato:  (0043) = 0F
            (0044) = 5C
            (0045) = 2B
cioè                 29.295 x 97 = 2.841.615
```

4) Divisione Binaria Con Segno

Scopo: Dividere il numero con segno a 16 bit contenuto nelle celle di memoria 0040 e 0041 (i MSB in 0041) per il numero con segno ad 8 bit contenuto nella cella 0042. I numeri sono normalizzati in modo che la grandezza della cella di memoria 0042 sia maggiore della cella 0041. Memorizzare il quoziente (con segno) nella cella 0043 ed il resto (sempre positivo) nella cella 0044.

Problemi Campione:

a.		(0040)	=	C0
		(0041)	=	FF (-64)
		(0042)	=	08
	Risultato:	(0043)	=	F8 (-8) quoziente
		(0044)	=	00 (0) remainder
b.		(0040)	=	93
		(0041)	=	ED (-4717)
		(0042)	=	47 (71 decimale)
	Risultato:	(0043)	=	BD (-67 decimale)
		(0044)	=	28 (+40 decimale)

Suggerimento: Determinare il segno del risultato, eseguire una divisione senza segno, e adattare il quoziente ed il resto in modo appropriato.

5) Numeri di Autocontrollo «Aligned 1, 3, 7 Mod 10»

Scopo: Calcolare una cifra di controllo di somma (checksum) da una stringa di cifre BCD. La lunghezza della stringa di cifre è contenuta nella cella di memoria 0041; la stringa di cifre (2 cifre BCD a parola) inizia nella cella 0042. Calcolare la cifra di controllo di somma col metodo «Aligned 1, 3, 7 Mod 10» e memorizzarla nella cella di memoria 0040.

La tecnica «Aligned 1, 3, 7 Mod 10» funziona come segue:

1) Azzerare il controllo di somma per iniziare.
2) Sommare la cifra più significativa al controllo di somma.
3) Moltiplicare la cifra necessaria per 3 e sommare il risultato al controllo di somma.
4) Moltiplicare la cifra successiva per 7 e sommare il risultato al controllo di somma.
5) Continuare il processo (Passi 2-4) finché non siano utilizzate tutte le cifre.
6) La cifra di controllo di somma è la cifra meno significativa del controllo di somma.

Per esempio, se la stringa di cifre è:

549321

il risultato sarà:

Controllo di somma = $5 + 3 \times 4 + 7 \times 9 + 3 + 3 \times 2 + 7 \times 1 = 96$
Cifra di autocontrollo = 6

Problemi Campione:

a. (0041) = 03
(0042) = 36
(0043) = 68
(0044) = 51
Risultato: controllo somma = $3 + 3 \times 6 + 7 \times 6 + 8 + 3 \times 5 + 7 \times 1 = 93$

b. (0040) = 03
(0041) = 04
(0042) = 50
(0043) = 29
(0044) = 16
(0045) = 83
Risultato: controllo somma = $5 + 3 \times 0 + 7 \times 2 + 9 + 3 \times 1 + 7 \times 6 + 8 + 3 \times 3 = 90$
(0040) = 00

Suggerimento: Notate che $7 = 2 \times 3 + 1$ e $3 = 2 \times 1 + 1$, così la formula $M_i = 2 \times M_{i-1} + 1$ può essere usata per calcolare il fattore di moltiplicazione successivo.

BIBLIOGRAFIA

1. Diversi algoritmi per la moltiplicazione sono descritti in T. Dollhoff, "Microprocessor Software: How To Optimize Timing and Memory Usage, Part Four, Techniques fort the Zilog 80", Digital Design, Febbraio 1977, pagg. 44-51.
2. Alcuni microprocessori (come il 9900, 8086 e Z-8000) posseggono istruzioni di moltiplicazione in hardware che sono alquanto più veloci, ma la massima velocità richiede l'aggiunta di hardware esterno. Altri metodi per implementare la moltiplicazione, la divisione, ed altre operazioni aritmetiche sono trattate in:

   Geist, D. J., "MOS Processor Picks up Speed with Bipolar Multipliers," Electronics, July 7, 1977, pp. 113-115.

   Kolodzinski, A. and D. Wainland, "Multiplying with a Microcomputer," Electronic Design, January 18, 1978, pp. 78-83.

   Mick, J. R. and J. Springer, "Single-chip Multiplier Expands Digital Role in Signal Processing," Electronics, May 13, 1976, pp. 103-108.

   Parasuraman, B., "Hardware Multiplication Techniques for Microprocessor Systems," Computer Design, April 1977, pp. 75-82.

   Tao, T. F. et al., "Applications of Microprocessors in Control Problems," 1977 Joint Automatic Control Conference Proceedings, San Francisco, CA., June 22-24, 1977.

   Waser, S., "State-of-the-art in High-Speed Arithmetic Integrated Circuits," Computer Design, July 1978, pp. 67-75.

   Weissberger, A. J. and T. Toal, "Tough Mathematical Tasks Are Child's Play for Number Cruncher," Electronics, February 17, 1977, pp. 102-107.
3. See J.R. Heir, "Self-Checking Number Sustems", Computer Design, Giugno 1974, pagg. 85-91.

Capitolo 9

TABELLE E LISTE

Le tabelle e le liste sono due delle strutture dati fondamentali usate con tutte i computer. Abbiamo già visto delle tabelle utilizzate per realizzare conversioni di codice e operazioni aritmetiche. Le tabelle possono anche essere adoperate per identificare o per rispondere a comandi e istruzioni, dati lineari, per fornire l'accesso a file o record, per definire il significato di tasti o interruttori, e per effettuare una scelta tra differenti programmi. Le liste sono solitamente meno strutturate delle tabelle. Esse possono registrare dei compiti da realizzare da parte del processore, messaggi oppure dati che il processore deve ricordare, oppure condizioni che sono variate o debbono essere controllate. Le tabelle costituiscono un metodo semplice per creare delle decisioni o dei problemi risolutivi, dato che non sono indispensabili dei calcoli o delle funzioni logiche. Il compito quindi si riduce all'organizzazione della tabella in modo che il particolare ingresso sia semplice da rintracciare. Le liste permettono l'esecuzione di sequenze di compiti, la preparazione di serie di risultati, e la costruzione di file di dati in relazione tra di loro (o data base). I problemi contemplano il metodo per aggiungere elementi da una lista e per trasferire da essa gli stessi.

ESEMPI

Aggiunta di un Ingresso ad una Lista

Scopo: Aggiungere il contenuto della cella di memoria 0040 ad una lista se non è già presente nella lista stessa. La lunghezza della lista è contenuta nella cella di memoria 0042.

Problemi Campione:

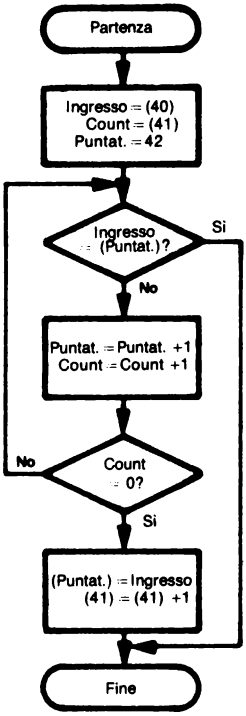
a.
```
          (0040) = 6B
          (0041) = 04
          (0042) = 37
          (0043) = 61
          (0044) = 38
          (0045) = 1D
Risultato: (0041) = 05
          (0046) = 6B
```

L'ingresso viene aggiunto alla lista, dato che non è già presente. La lunghezza della lista viene aumentata di 1.

b.
```
          (0040) = 6B
          (0041) = 04
          (0042) = 37
          (0043) = 6B
          (0044) = 38
          (0045) = 1D
```

Risultato: Nessuna variazione, poiché l'ingresso è già nella lista

Diagramma di Flusso:



Programma Sorgente:

```
        LD      HL,40H    ;PUNTA AD INGRESSO
        LD      A,(HL)    ;PRENDI INGRESSO
        INC     HL        ;PUNTA A COUNT
        LD      B,(HL)    ;COUNT = LUNGHEZZA DI LISTA
        INC     HL        ;PUNTA ALL'INIZIO DELLA LISTA
SRLST:  CP      (HL)      ;INGRESSO = ELEMENTO DELLA LISTA?
        JR      Z,DONE    ;SÌ, VAI AVANTI
        INC     HL        ;NO, VAI ALL'ELEMENTO SUCCESSIVO
        DJNZ    SRLST
        LD      (HL),A    ;SOMMA INGRESSO ALLA LISTA
        LD      HL,41H    ;SOMMA 1 ALLA LUNGHEZZA DI LISTA
        INC     (HL)
DONE:   HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)		
0000	21		LD	HL,40H
0001	40			
0002	00			
0003	7E		LD	A,(HL)
0004	23		INC	HL
0005	46		LD	B,(HL)
0006	23		INC	HL
0007	BE	SRLST:	CP	(HL)
0008	28		JR	Z,DONE
0009	08			
000A	23		INC	HL
000B	10		DJNZ	SRLST
000C	FA			
000D	77	ADELM:	LD	(HL),A
000E	21		LD	HL,41H
000F	41			
0010	00			
0011	34		INC	(HL)
0012	76	DONE:	HALT	

Potremmo anche utilizzare nel vostro esempio l'istruzione di ricerca nel blocco CPIR, come segue:

Programma Sorgente:

```
        LD      HL,40H   ;PUNTA AD INGRESSO
        LD      A,(HL)   ;PRENDI INGRESSO
        INC     HL       ;PUNTA A COUNT
        LD      B,0      ;COUNT = LUNGHEZZA DI LISTA (16 BITS)
        LD      C,(HL)
        INC     HL       ;PUNTA ALL'INIZIO DI LISTA
        CPIR             ;RICERCA INGRESSO NELLA LISTA
        JR      Z,DONE   ;VAI A DONE SE SI È TROVATO INGRESSO
        LD      (HL),A   ;ALTRIMENTI SOMMA INGRESSO ALLA LISTA
        LD      HL,41H   ;SOMMA 1 ALLA LUNGHEZZA DI LISTA
DONE:   HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)	
0000	21	LD	HL,40H
0001	40		
0002	00		
0003	7E	LD	A,(HL)
0004	23	INC	HL
0005	06	LD	B,0
0006	00		
0007	4E	LD	C,(HL)
0008	23	INC	HL
0009	ED	CPIR	
000A	B1		
000B	28	JR	Z,DONE
000C	05		
000D	77	LD	(HL),A
000E	21	LD	HL,41H
000F	41		
0010	00		
0011	34	INC	(HL)
0012	76	DONE: HALT	

Ricordate che CPIR ripete automaticamente l'istruzione di Ricerca fondamentale finché BC decrementandosi raggiunge il valore zero oppure il confronto ha un esito positivo (vale a dire, A = (HL)).

Fate attenzione alle seguenti lievi differenze rispetto alla precedente versione:

1) BC è un contatore a 16 bit. Perciò, CPIR può trattare stringhe più lunghe di 256 byte.
2) Il bit di Parità/Overflow (P/O) viene azzerato se BC raggiunge nei suoi decrementi lo zero, altrimenti viene posizionato ad uno.

HASHING

Chiaramente, questo metodo di aggiunta di elementi è inefficiente se la lista è lunga. Si potrebbe perfezionare la procedura limitando la ricerca ad una parte della lista oppure ordinando quest'ultima. Si potrebbe limitare la ricerca usando l'ingresso per ottenere un punto di partenza nella lista. Questo metodo è chiamato «hashing» ed è molto simile alla selezione di una pagina iniziale in un dizionario o in una guida in base alla prima lettera in un ingresso. Si potrebbe porre in ordine la lista secondo il valore numerico. La ricerca potrebbe quindi terminare allorquando i valori della lista vadano al di là dell'ingresso (maggiori o minore, dipendono dalla tecnica d'ordinamento utilizzata). Un nuovo ingresso dovrebbe essere inserito in modo corretto, e tutti gli altri ingressi dovrebbero essere spostati verso il basso nella lista.

Il programma potrebbe essere ristrutturato per usare due tabelle. Una di queste potrebbe fornire un punto di partenza nell'altra tabella; per esempio, il punto di ricerca potrebbe essere basato sulla cifra a 4 bit più o meno significativa nell'ingresso.

Il programma non funziona se la lunghezza della lista potesse essere zero (cosa accade?). Si potrebbe evitare questo problema controllando inizialmente la lunghezza. La procedura d'inizializzazione per il primo programma dovrebbe quindi essere:

```
        LD      HL,40H   ;PUNTA AD INGRESSO
        LD      A,(HL)   ;PRENDI INGRESSO
        INC     HL       ;PUNTA A LUNGHEZZA
        LD      B,(HL)   ;COUNT = LUNGHEZZA DI LISTA
        INC     HL       ;PUNTA ALL'INIZIO DELLA LISTA
        INC     B        ;COUNT È ZERO?
        DEC     B
        JR      Z,ADELM  ;SÌ, VAI A SOMMARE INGRESSO ALLA LISTA
        •
        •
        •
ADELM   LD      (HL),A   ;SOMMA INGRESSO ALLA LISTA
```

Notate che la sequenza INC, DEC è un modo semplice per verificare la presenza del valore zero in un registro senza l'uso dell'Accumulatore o senza alcuna variazione del valore nel registro.

La procedura:

```
        LD      HL,ADDR
        INC     (HL)
```

è un modo rapido per aggiungere 1 ad un contatore contenuto nella locazione di memoria ADDR senza usare l'Accumulatore. Si può far uso di DEC (HL) in maniera simile per sottrarre 1 dal contatore. LD (HL), CONST può caricare un valore di partenza (come uno zero) nel contatore. Le celle di memoria dovrebbero, naturalmente, essere usate come contatori solamente quando non sono prontamente a disposizione dei registri accessibili.

Se ciascun ingresso fosse più lungo di una parola, sarebbe necessario un programma di confronto di struttura. Il programma dovrebbe proseguire verso il successivo ingresso se viene a mancare una eguaglianza; cioè scavalcare l'ultima parte dell'ingresso corrente una volta che è stata scoperta una disiguaglianza.

Controllo di una Lista Ordinata

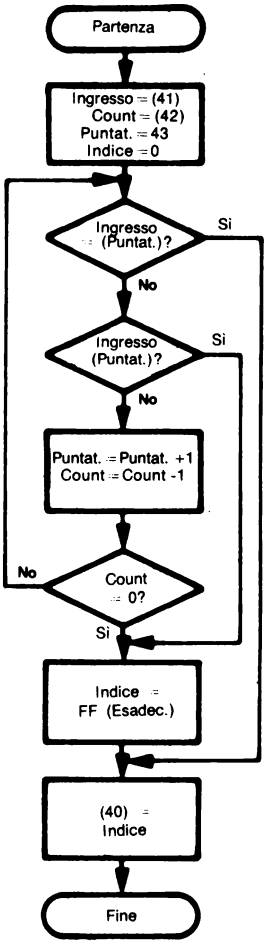
Scopo: Controllare il contenuto della cella di memoria 0041 per vedere se è una lista ordinata. La lunghezza della lista è posta nella cella 0042; la lista stessa inizia nella cella 0043 e consiste di numeri binari privi di segno disposti in ordine crescente. Se il contenuto della cella 0041 è nella lista, azzerare la cella di memoria 0040, altrimenti posizionarla al valore FF (esadecimale).

Problemi Campione:

a.
(0041) = 6B
(0042) = 04
(0043) = 37
(0044) = 55
(0045) = 7D
(0046) = A1
Risultato: (0040) = FF poiché 6B non è nella lista

b.
(0041) = 6B
(0042) = 04
(0043) = 37
(0044) = 55
(0045) = 6B
(0046) = A1
Risultato: (0040) = 00, poiché 6B è nella lista

Diagramma di Flusso:



Il processo di ricerca è qui un po' diverso dato che gli elementi sono posti in ordine. Una volta che si trova un elemento maggiore dell'ingresso, la ricerca ha termine, dato che gli elementi successivi saranno ancor maggiori. Potreste sperimentare un esempio per convincervi che la procedura funziona.

METODI DI RICERCA

Come nel precedente problema, una tabella oppure un altro metodo che potesse scegliere un buon punto di partenza accelererebbe la ricerca. Un altro metodo sarebbe quello di iniziare nel mezzo e determinare in quale metà della lista era compreso l'ingresso, quindi dividere la metà in altre due metà, ecc. Questo metodo è detto di ricerca binaria, dato che divide ogni volta la parte restante della lista a metà.

Programma Sorgente:

```
        LD      HL,41H   ;PUNTA AD INGRESSO
        LD      A,(HL)   ;PRENDI INGRESSO
        INC     HL       ;PUNTA A LUNGHEZZA
        LD      B,(HL)   ;COUNT = LUNGHEZZA DI LISTA
        LD      C,0      ;INDICE = ZERO PER ELEMENTI IN LISTA
        INC     HL       ;PUNTA ALL'INIZIO DI LISTA
SRLST:  CP      (HL)     ;INGRESSO = ELEMENTO DELLA LISTA?
        JR      Z,DONE   ;SÌ, COMPLETAMENTO DELLA RICERCA
        JR      C,NOTIN  ;INGRESSO NON IN LISTA SE MINORE DELL'ELEMENTO
        INC     HL
        DJNZ    SRLST
NOTIN:  LD      C,OFFH   ;INDICE = FF PER ELEMENTO NON IN LISTA
DONE:   LD      A,C      ;SALVA INDICE
        LD      (40H),A
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)		
0000	21		LD	HL,41H
0001	41			
0002	00			
0003	7E		LD	A,(HL)
0004	23		INC	HL
0005	46		LD	B,(HL)
0006	0E		LD	C,0
0007	00			
0008	23		INC	HL
0009	BE	SRLST:	CP	(HL)
000A	28		JR	Z,DONE
000B	07			
000C	38		JR	C,NOTIN
000D	03			
000E	23		INC	HL
000F	10		DJNZ	SRLST
0010	F8			
0011	0E	NOTIN:	LD	C,OFFH
0012	FF			
0013	79		LD	A,C
0014	32		LD	(40H),A
0015	40			
0016	00			
0017	76		HALT	

Le istruzioni dello Z80 di ricerca nel blocco non sono utili qui come nell'esempio precedente poiché si vuol realizzare più di una semplice ricerca. Ora si vuole anche controllare per vedere se si è esaminata la parte attinente della lista (cioè la parte dove gli elementi sono minori o uguali all'ingresso). Provate a riscrivere il programma per usare CPI. Ricordate che dovete usare il flag di Parità/Overflow per determinare se il contatore di byte ha raggiunto, decrementandosi, il valore zero.

Rimozione di un Elemento da una Fila

Scopo: Le locazioni di memoria 0042 e 0043 contengono l'indirizzo dell'inizio della fila (MSB in 0043). Porre l'indirizzo del primo elemento (inizio) di una fila nelle celle di memoria 0040 e 0041 (MSB in 0041) e aggiornare la fila per trasferire l'elemento. Ciascun elemento nella fila è lungo due byte e contiene l'indirizzo del successivo elemento a due byte nella fila. L'ultimo elemento nella fila contiene zero per indicare che non c'è subito di seguito alcun elemento.

Le file vengono usate per memorizzare dati nell'ordine nel quale saranno poi utilizzati, oppure compiti nell'ordine nel quale verranno eseguiti. La fila è una struttura dati di tipo «first-in, first-out»; cioè gli elementi vengono rimossi dalla fila nello stesso ordine nel quale erano entrati. I sistemi operativi pongono i compiti nelle file in modo tale che essi saranno eseguiti nel giusto ordine. I driver dell'I/O trasferiscono i dati da e verso le file in modo che essi saranno trasmessi o manipolati nel giusto ordine. I buffer possono essere posti in fila in modo che quello di seguito disponibile possa facilmente essere individuato e quelli che vengono ceduti possano essere aggiunti alla memorizzazione disponibile. Le file possono anche essere usate per collegare richieste di memorizzazione, temporizzazione, oppure I/O in modo che possano essere soddisfatte nel corretto ordine.

Nelle reali applicazioni, ogni elemento nella fila conterrà tipicamente una grossa quantità d'informazioni o di spazio per memorizzazione oltre all'indirizzo necessario per collegare l'elemento a quello successivo.

Problemi Campione:

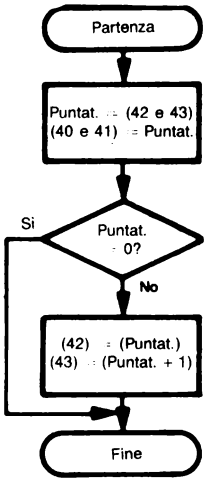
a.

	(0042)	=	46	indirizzo del primo elemento in coda
	(0043)	=	00	
	(0046)	=	4D	indirizzo del secondo elemento in coda
	(0047)	=	00	
	(004D)	=	00	fine della coda
	(004E)	=	00	
Risultato:	(0040)	=	46	indirizzo dell'elemento spostato dalla coda
	(0041)	=	00	
	(0042)	=	4D	indirizzo del nuovo primo elemento della coda
	(0043)	=	00	

b.

	(0042)	=	00	coda vuota
	(0043)	=	00	
Risultato:	(0040)	=	00	nessun elemento disponibile dalla coda
	(0041)	=	00	

Diagramma di Flusso:



Programma Sorgente:

```
        LD      HL,(42H) ;PRENDI L'INDIRIZZO DI TESTA DELLA CODA
        LD      (40H),HL ;SOSTA LA TESTA DELLA CODA
        LD      A,H      ;LA CODA È VUOTA?
        OR      L
        JR      Z,DONE   ;SÌ, VAI A DONE
        LD      E,(HL)   ;NO, PRENDI L'INDIRIZZO DELL'ELEMENTO SUCCESSIVO
        INC     HL
        LD      D,(HL)
        LD      (42H),DE ;SPOSTA L'ELEMENTO SUCCESSIVO IN TESTA ALLA CODA
DONE:   HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)	
0000	2A	LD	HL,(42H)
0001	42		
0002	00		
0003	22	LD	(40H),HL
0004	40		
0005	00		
0006	7C	LD	A,H
0007	B5	OR	L
0008	28	JR	Z,DONE
0009	07		
000A	5E	LD	E,(HL)
000B	23	INC	HL
000C	56	LD	D,(HL)
000D	ED	LD	(42H),DE
000E	53		
000F	42		
0010	00		
0011	76	DONE: HALT	

Questo sistema che utilizza la fila può manipolare liste che non sono in locazioni di memoria sequenzialmente adiacenti. Ogni elemento deve contenere l'indirizzo dell'elemento seguente. Tali liste permettono di trattare dati o compiti nell'esatto ordine, di cambiare variabili, o di inserire definizioni in un programma. È necessaria un'ulteriore memorizzazione, ma gli elementi possono facilmente essere aggiunti alla fila oppure cancellati dalla stessa.

Notate l'uso della sequenza:

```
LD      A,H
OR      L
```

per determinare se il contenuto della coppia di registri a 16 bit è uguale a zero. Ricordate che INC e DEC non agiscono su nessun flag se applicate ad una coppia di registri. Provate a escogitare qualche altra sequenza che potrebbe trattare questo problema – ciò ovviamente avviene ogni volta che si utilizza un contatore a 16 bit anziché uno ad 8 bit il quale è stato adoperato nella maggior parte degli esempi.

Un problema è che non ci sono istruzioni che caricano una coppia di registri utilizzando l'indirizzo contenuto in una coppia di registri. È necessaria una sequenza di istruzioni ogni volta che una coppia di registri deve essere caricata direttamente.

Potrebbe essere utile mantenere i puntatori rivolti verso ambedue le estremità della fila invece che solo verso il suo inizio. La struttura dei dati può quindi essere usata o in modo «first-in», first-out» o in modo «last-in, first-out», a seconda se i nuovi elementi vengono aggiunti all'inizio o in coda. Come varieresteste l'esempio di programma in modo che le locazioni di memoria 0044 e 0045 contengano l'indirizzo dell'ultimo elemento (coda) della fila?

Se non vi sono elementi nella fila, il programma azzera le celle di memoria 0040 e 0041. Un programma che ha richiesto un elemento dalla fila dovrebbe quindi controllare quelle celle di memoria per vedere se la sua richiesta è stata soddisfatta. Siete in grado di suggerire altri modi per fornire queste informazioni?

Classificazione di Parole ad 8 Bit

Scopo: Classificare una serie di numeri binari privi di segno in ordine decrescente. La lunghezza della serie è contenuta nella locazione di memoria 0040 e la serie stessa inizia nella locazione 0041.

Problema Campione:

```
          (0040) = 06
          (0041) = 2A
          (0042) = B5
          (0043) = 60
          (0044) = 3F
          (0045) = D1
          (0046) = 19

Risultato: (0041) = D1
           (0042) = B5
           (0043) = 60
           (0044) = 3F
           (0045) = 2A
           (0046) = 19
```

SEMPLICE ALGORITMO DI CLASSIFICAZIONE

Una semplice tecnica di classificazione funziona come segue:

Fase 1) Azzerare un flag INTER
Fase 2) Esaminare ciascuna coppia consecutiva di numeri nella serie. Se ve ne sono alcuni che non sono in ordine, scambiarli e posizionare ad 1 INTER.
Fase 3) Se Inter = 1 dopo che è stata esaminata l'intera serie, ritornare alla Fase 1.

INTER sarà posizionato ad 1 se una qualsiasi coppia consecutiva di numeri non è in ordine. Pertanto, se INTER = 0 al termine di un passaggio lungo l'intera seria, questa è nell'esatto ordine.

Questo metodo di classificazione è nota come «bubble sort». Esso è un facile algoritmo da implementare. Tuttavia, dovrebbero essere considerate altre tecniche di classificazione quando si tratta di lunghe liste dove è importante la velocità.[2]

La tecnica funziona come segue in un semplice caso. Supponiamo di voler classificare una serie in ordine decrescente; la serie è composta di quattro elementi — 12, 03, 15, 08.

1°) Ripetizione:

Fase 1) INTER = 0
Fase 2) L'ordine finale della serie è:
12
15
08
03
dato che la seconda coppia (03, 15) viene scambiata e così è la terza coppia (03, 08).
INTER = 1

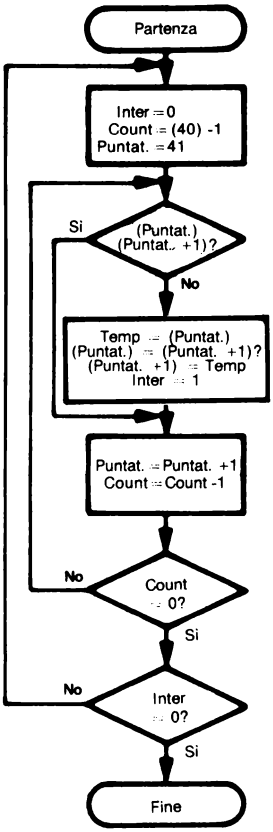
2°) Ripetizione:

Fase 1) INTER = 0
Fase 2) L'ordine finale della serie è:
15
12
08
03
dato che la prima coppia (12, 15) è scambiata. INTER = 1.

3°) Ripetizione:

Fase 1) INTER = 0
Fase 2) Gli elementi sono già in ordine, così non sono necessari scambi e INTER rimane zero.

Diagramma di Flusso:



Programma Sorgente:

```
SORT:   LD      C,0         ;AZZERA IL FLAG DI SCAMBIO
        LD      HL,40H      ;COUNT = LUNGHEZZA DELL'ARRAY
        LD      B,(HL)
        DEC     B           ;NUMERO DI COPPIE = COUNT—1
        INC     HL          ;PUNTA ALL'INIZO DELL'ARRY
PASS1:  LD      A,(HL)      ;PRENDI L'ELEMENTO DALL'ARRAY
        INC     HL
        CP      (HL)        ;È MINORE DELL'ELEMENTO SUCCESSIVO?
        JR      NC,CNT      ;NO, NON È NECESSARIO NESSUN SCAMBIO
        LD      D,(HL)      ;SÌ, SCAMBIA GLI ELEMENTI
        LD      (HL),A
        DEC     HL
        LD      (HL),D
        INC     HL
        LD      C,1         ;POSIZIONA AD 1 IL FLAG DI SCAMBIO
CNT:    DJNZ    PASS1
        DEC     C           ;IL FLAG DI SCAMBIO È STATO POSIZIONATO?
        JR      Z,SORT      ;SÌ, FAI UN ALTRO PASSO
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)		
0000	0E	SORT:	LD	C,0
0001	00			
0002	21		LD	HL,40H
0003	40			
0004	00			
0005	46		LD	B,(HL)
0006	05		DEC	B
0007	23		INC	HL
0008	7E	PASS1:	LD	A,(HL)
0009	23		INC	HL
000A	BE		CP	(HL)
000B	30		JR	NC,CNT
000C	07			
000D	56		LD	D,(HL)
000E	77		LD	(HL),A
000F	2B		DEC	HL
0010	72		LD	(HL),D
0011	23		INC	HL
0012	0E		LD	C,1
0013	01			
0014	10	CNT:	DJNZ	PASS1
0015	F2			
0016	0D		DEC	C
0017	28		JR	Z,SORT
0018	E7			
0019	76		HALT	

Il caso dove due elementi nella serie sono eguali è qui molto importante. Il programma non eseguirebbe uno scambio in quel caso, dato che lo scambio avverrebbe ad ogni passaggio. Il risultato sarebbe che ad ogni passaggio si posizionorebbe ad 1 il flag di scambio, provocando così un ciclo interminabile.

Il programma deve ridurre il contatore di 1, dato che il numero di coppie consecutive è inferiore di uno rispetto al numero di elementi (l'ultimo elemento non ne ha altri che lo seguono). Prima di iniziare ciascun passaggio di classificazione, ci si deve ricordare di inizializzare di nuovo il contatore, il puntatore ed il flag di scambio.

Vi sono molte possibili variazioni secondarie da operare su questo programma. Per esempio, potremmo usare RES 0,C e SET 0,C per azzerare e posizionare ad 1 il flag di scambio invece di LD C,0 e LD C,1. Potremmo inoltre utilizzare la sequenza MOV B,C seguita da DJNZ SORT per controllare il flag di scambio.

Notate che il Registro B dovrebbe essere adoperato come contatore interno, dato che questo contatore viene decrementato molto spesso. Ciò permette di sfruttare pienamente l'istruzione DJNZ.

L'indirizzamento indicizzato sarebbe un modo conveniente per realizzare lo scambio se i registri indice dello Z80 fossero più accessibili. Provate a riscrivere il programma in modo tale da usare uno dei registri indice e confrontate il tempo di esecuzione e l'impiego di memoria del programma riscritto rispetto a quelli del programma originale.

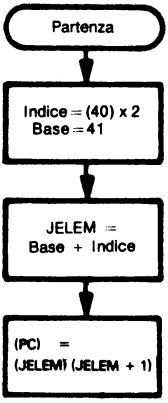
Utilizzazione di una Tabella di Salto Ordinata

Scopo: Utilizzare il contenuto della cella di memoria 0040 come indice di una tabella di salto che ha inizio nella locazione di memoria 0041. Ciascun ingresso nella tabella di salto contiene un indirizzo di 16 bit con i LSB nella prima parola. Il programma dovrebbe trasferire il controllo all'indirizzo con l'indice appropriato: cioè, se l'indice è 6, il programma salta all'indirizzo ingresso # 6 nella tabella. Supporre che la tabella abbia meno di 128 ingressi.

Problema Campione:

(0040)	=	02
(0041)	=	48
(0042)	=	00
(0043)	=	4C
(0044)	=	00
(0045)	=	50
(0046)	=	00
(0047)	=	54
(0048	=	00
Risultato: (PC)	=	0050, poiché è l'ingresso = 2, (a partire da zero) nella tabella dei salti.

Diagramma di Flusso:



L'ultimo blocco ha come conseguenza un trasferimento del controllo all'indirizzo ottenuto dalla tabella.

Programma Sorgente:

```
LD      HL,40H   ;PUNTA AD INDICE
LD      A,(HL)   ;PRENDI INDICE
ADD     A,A      ;RADDOPPIA INDICE PER LA TABELLA A 2 BYTE
LD      E,A
LD      D,O      ;ESPANDI INDICE A 16 BIT
INC     HL       ;INDIRIZZO BASE DELLA TABELLA DEI SALTI
ADD     HL,DE    ;INDICE NELLA TABELLA DEI SALTI
LD      E,(HL)   ;PRENDI I BIT LSB DELL'INDIRIZZO DI DESTINAZIONE
INC     HL
LD      D,(HL)   ;PRENDI I BIT MSB DELL'INDIRIZZO DI DESTINAZIONE
EX      DE,HL
JP      (HL)     ;TRASFERISCI IL CONTROLLO ALLA DESTINAZIONE
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice mnemonico)	
0000	21	LD	HL,40H
0001	40		
0002	00		
0003	7E	LD	A,(HL)
0004	87	ADD	A,A
0005	5F	LD	E,A
0006	16	LD	D,0
0007	00		
0008	23	INC	HL
0009	19	ADD	HL,DE
000A	5E	LD	E,(HL)
000B	23	INC	HL
000C	56	LD	D,(HL)
000D	EB	EX	DE,HL
000E	E9	JP	(HL)

Le tabelle di salto sono molto utili in situazioni dove deve essere scelta una diversa «routine». Queste situazioni nascono nella decodifica di comandi nella selezione di programmi di prova, nella scelta di metodi alternativi, o nella selezione di una configurazione per l'I/O.

La tabella di salto sostituisce un'intera serie di operazioni di salto condizionato. Il programma che accede alla tabella di salto potrebbe essere usato per accedere a parecchie tabelle diverse variando semplicemente l'indirizzo di partenza.[3]

Il dato deve essere moltiplicato per due per fornire l'indice esatto, dato che ciascun ingresso nella tabella di salto è un indirizzo a due byte.

SALTI INDIRETTI

L'istruzione JP (HL), che trasferisce il contenuto della Coppia di Registri HL nel Contatore del Programma, è un salto indiretto che è assai comodo nelle tabelle di salto e nei programmi monitor. Notate che JP (HL) è una istruzione di Salto, poiché essa pone un nuovo contenuto nel Contatore del Programma; comunque essa permette di porre un indirizzo variabile direttamente nel Contatore del Programma. Tutte le istruzioni di Salto Condizionato (e le istruzioni di Call) fanno uso di indirizzi fissi. Le sole istruzioni di Salto con una flessibilità simile sono le istruzioni composte da due parole JP (IX) e JP (JY).

Non è necessaria alcuna operazione finale, dato che JP (HL) trasferisce il controllo all'indirizzo ottenuto dalla tabella di salto.

PROBLEMI

1) Rimozione di un Ingresso Da una Lista

Scopo: Togliere il contenuto della cella di memoria 0040 da una lista se vi è presente. La lunghezza della lista è contenuta nella cella 0041 e la stessa lista inizia nella cella 0042. Spostare gli ingressi di una posizione sotto quello eliminato e diminuire la lunghezza della lista una unità.

Problemi Campione:

a.

(0040) = 6B
(0041) = 04
(0042) = 37
(0043) = 61
(0044) = 28
(0045) = 1D

Risultato: Nessuna variazione, dato che l'ingresso non è presente nella lista

b

(0040) = 6B
(0041) = 04
(0042) = 37
(0043) = 6B
(0044) = 28
(0045) = 1D

Risultato: (0041) = 03
(0042) = 37
(0043) = 28
(0044) = 1D

L'ingresso viene tolto dalla lista e quelli sotto di esso sono spostati in su di una posizione. La lunghezza della lista viene diminuita di 1.

2) Aggiunta di un Ingresso ad una Lista Ordinata

Scopo: Porre il contenuto della locazione di memoria 0040 in una lista ordinata se non vi è già presente. La lunghezza della lista è contenuta nella cella di memoria 0041 ed essa stessa inizia nella cella 0042; la lista consiste di numeri binari in ordine crescente. Porre il nuovo ingresso nella posizione corretta all'interno della lista, spostare in giù gli elementi che si trovano sotto di esso, ed aumentare di 1 la lunghezza della lista.

Problemi Campione:

a.		(0040)	= 6B
		(0041)	= 04
		(0042)	= 37
		(0043)	= 55
		(0044)	= 7D
		(0045)	= A1
	Risultato:	(0041)	= 05
		(0044)	= 6B
		(0045)	= 7D
		(0046)	= A1
b.		(0040)	= 6B
		(0041)	= 04
		(0042)	= 37
		(0043)	= 55
		(0044)	= 6B
		(0045)	= A1

Risultato: Nessun cambiamento, poiché il dato in ingresso è già nella lista.

3) Aggiunta di un Elemento ad una Fila

Scopo: Aggiungere l'indirizzo presente nelle celle di memoria 0040 e 0041 (MSB in 0041) ad una fila. L'indirizzo del primo elemento della fila è nelle celle di memoria 0042 e 0043 (MSB in 0043). Ogni elemento nella fila contiene o l'indirizzo dell'elemento che lo segue nella fila o il valore zero se non c'è nessun elemento di seguito; tutti gli indirizzi sono a 16 bit con i bit più significativi nella seconda parola dell'elemento. Il nuovo elemento va al termine (coda) della fila; il suo indirizzo sarà nell'elemento che era al termine della fila e conterrà zero per indicare che esso è ora in fondo alla fila.

Problema Campione:

	(0040)	= 4D	nuovo elemento da aggiungere alla coda
	(0041)	= 00	
	(0042)	= 46	puntatore verso la testa della coda
	(0043)	= 00	
	(0046)	= 00	ultimo elemento della coda
	(0047)	= 00	
Risultato:	(0046)	= 4D	ultimo vecchio elemento che punta al nuovo ultimo elemento
	(0047)	= 00	
	(004D)	= 00	nuovo ultimo elemento della coda
	(004E)	= 00	

Come fareste per aggiungere un elemento alla fila se le locazioni di memoria 0044 e 0045 contenessero l'indirizzo della coda della fila?

4) Classificazione di Parola a 16 Bit

Scopo: Classificare una serie di numeri binari a 16 bit privi di segno in ordine decrescente. La lunghezza della serie è contenuta nella locazione di memoria 0040 e la serie stessa inizia nella locazione 0041. Ciascun numero a 16 bit è memorizzato con i bit meno significativi nella prima parola.

Problemi Campione:

	(0040)	=	03
	(0041)	=	D1
	(0042)	=	19
	(0043)	=	60
	(0044)	=	3F
	(0045)	=	2A
	(0046)	=	B5
Risultato:	(0041)	=	2A
	(0042)	=	B5
	(0043)	=	60
	(0044)	=	3F
	(0045)	=	D1
	(0046)	=	19

I numeri sono B52A, 3F60, e 19D1.

5) Utilizzazione di una Tabella di Salto con una Chiave

Scopo: Usare il contenuto della cella di memoria 0040 come chiave di una tabella di salto che inizia nella cella di memoria 0041. Ciascun ingresso nella tabella di salto contiene un valore chiave ad 8 bit seguito da un indirizzo di 16 bit (MSB nella seconda parola) al quale il programma dovrebbe trasferire il controllo se la chiave è uguale a quel particolare valore chiave.

Problema Campione:

	(0040)	=	38
	(0041)	=	32
	(0042)	=	4B
	(0043)	=	00
	(0044)	=	35
	(0045)	=	4D
	(0046)	=	00
	(0047)	=	38
	(0048)	=	4F
	(0049)	=	00
Risultato:	(PC)	=	004F, dato che quell'indirizzo corrisponde al valore chiave 38.

Provate a riscrivere il programma con e senza l'istruzione CPIR. Potete pensare ad un modo per semplificare la versione che utilizza l'istruzione CPIR? Suggerimento: porre tutte le parole ad 8 bit corrispondenti in tabelle separate in modo che il programma abbia solo da aggiungere 1 al puntatore della tabella per spostarsi da un valore chiave al successivo.[4]

BIBLIOGRAFIA

1. Knuth descrive altre tecniche di ricerca nel suo libro The Art of Computer Programming, Volume III: Sorting and Searching, Addison-Wesley, Reading, Mass. 1978. Knuth ha inoltre trattato la ricerca e l'hashing in un modo più elementare in un articolo intitolato "Algorithms" (vedi l'edizione di Aprile 1977 di Scientific American).

2. Ci sono molti algoritmi di classificazione che variano ampiamente come efficacia. Knuth ne descrive alcuni nel libro su menzionato (The Art of Computer Programming, Volume III: Sorting and Searching). Gli algoritmi di classificazione e di ricerca sono anche trattati in K.A. Schember e J.R. Rumsey, "Minimal Storage Sorting and Searching Techniques for RAM Applications, a Tutorial", Computer, Giugno 1977, pagg. 92-100.

3. Ci sono ulteriori esempi dell'uso di tabelle di salto in L.A. Leventhal, "Cut Your Processor's Computation Time", Electronic Design, 16 Agosto 1977, pagg. 82-89, e nel Capitolo 7 di J.B. Peatman, Microcomputer-Based Design, Mc Graw-Hill, New York, 1977.

4. Questo metodo è trattato da T. Dollhoff in "Microprocessor Software: How to Optimize Timing and Memory Usage; Part Four: Tecniques for the Zilog Z80", Digital Design, Febbraio 1977, pagg. 48-49.

Capitolo 10

SUBROUTINE

Nessuno degli esempi che abbiamo finora rappresentato costituisce tipicamente di per se stesso un programma intero. La maggior parte dei programmi reali esegue una serie di compiti, molti dei quali possono essere gli stessi oppure comuni a parecchi programmi differenti. Abbiamo bisogno di un modo per formulare questi compiti una sola volta e rendere le formulazioni convenientemente disponibili sia in diversi punti del programma corrente che in altri programmi.

LIBRERIA DI SUBROUTINE

Il tipico metodo è quello di scrivere subroutine (sottoprogrammi) che eseguono compiti particolari. Le sequenze di istruzioni risultanti possono essere scritte una sola volta, una sola volta collaudate, e quindi utilizzate ripetutamente. Esse non possono formare una libreria delle subroutine che fornisce soluzioni documentate per problemi comuni.

ISTRUZIONI PER LE SUBROUTINE

La maggior parte dei microprocessori posseggono istruzioni particolari per il trasferimento del controllo a delle subroutine e per la restituzione del controllo stesso al programma principale. Ci si riferisce spesso all'istruzione particolare che trasferisce il controllo ad una subroutine come Call, Jump-to-Subroutine, Jump e Mark Space, oppure Jump e Link. L'istruzione speciale che richiede il controllo al programma principale è solitamente chiamata Ritorno. Sul microprocessore Z80, l'istruzione di Chiamata (Call) salva il vecchio valore del Contatore del Programma nell'area di Stack RAM prima di porre l'indirizzo di partenza della subroutine nel Contatore del Programma; l'istruzione di Ritorno (RET) prende il vecchio contenuto dallo Stack e lo rimette nel Contatore del Programma. L'effetto è quello di trasferire il controllo del programma, la prima volta alla subroutine e poi di nuovo al programma principale. Chiaramente la subroutine stessa può trasferire il controllo ad un'altra subroutine e così via.

Per essere effettivamente utile, una subroutine deve essere generica. Una routine che può eseguire solo un compito specifico, quale ad esempio la ricerca di una particolare lettera in una stringa d'ingresso di lunghezza fissata, non sarà di grande utilità. Se, d'altra parte, la subroutine potesse ricercare una qualunque lettera in stringhe di qualsiasi lunghezza, sarebbe assai più utile. Chiamiamo «parametri» i dati o gli indirizzi ai quali la subroutine permette di variare. Una parte importante nella scrittura delle subroutine riguarda la decisione su quali variabili dovrebbero costituire i parametri.

PASSAGGIO DI PARAMETRI

Un problema è la cessione dei parametri alla subroutine; questo processo viene detto passaggio dei parametri. Il metodo più semplice per il programma principale è quello di collocare i parametri nei registri. Quindi la subroutine può semplicemente presumere che i parametri siano lì. Naturalmente, questa tecnica viene limitata dal numero di registri che sono disponibili. I parametri possono, tuttavia, essere indirizzi oltre che dati. Per esempio, una routine di classificazione potrebbe cominciare con l'indirizzo di partenza di un contenuto nella Coppia di Registri HL.

Se ci sono più parametri altri metodi sono necessari. Un'opportunità è quella di usare lo Stack. Il programma principale può porre i parametri nello Stack e la subroutine ripristinarli. I vantaggi di questo metodo sono che lo Stack è praticamente illimitato nelle sue dimensioni, e che i dati nello Stack non vengono perduti anche se lo Stack viene usato di nuovo.

Gli svantaggi sono che poche istruzioni dello Z80 utilizzano lo Stack, e l'istruzione Call memorizza nello Stack anche l'indirizzo di ritorno. Un altro metodo è quello di usare un'area di memoria per i parametri. Il programma principale può porre l'indirizzo dell'area nella Coppia di Registri HL oppure in uno dei registri indice e la subroutine può ripristinare i dati secondo necessità. Tuttavia, questa procedura è poco efficace se gli stessi parametri sono indirizzi.

RIALLOCAZIONE

Spesso una subroutine deve avere particolari caratteristiche. Una subroutine è riallocabile se può essere collocata in qualsiasi punto della memoria. Si può utilizzare facilmente una tale subroutine, senza badare alla posizione di altri programmi o all'ordinamento della memoria. Un programma rigorosamente riallocabile può far a meno di utilizzare indirizzi assoluti; tutti gli indirizzi devono essere in relazione all'inizio del programma. È necessario un caricatore di riallocazione per collocare il programma in memoria correttamente; il caricatore darà inizio al programma dopo gli altri programmi e aggiungerà l'indirizzo di partenza o costante di riallocazione a tutti gli indirizzi nel programma.

SUBROUTINE RIENTRANTE

Una subroutine è rientrata se può essere interrotta e chiamata dal programma interrompente e se può tuttavia fornire i risultati esatti di ambedue i programmi interrompente e interrotto. Il rientramento è importante per le subroutine standard in un sistema basato sugli interrupt. Altrimenti le routine di servizio dell'interrupt non possono far uso delle subroutine standard senza causare errori. Le subroutine del microprocessore si prestano facilmente ad essere rientranti, dato che l'istruzione di Chiamata utilizza lo Stack e quella procedura è automaticamente rientrante. Il solo requisito restante è che la subroutine usi i registri e lo Stack invece di locazioni di memoria fisse come memorizzazioni temporanee. Ciò è piuttosto scomodo, ma si può solitamente eseguire se necessario.

Una subroutine è ripetitiva se essa chiama se stessa. Una tale subroutine deve chiaramente essere anche rientrante. Tuttavia, le subroutine ripetitive sono rare nelle applicazioni del microprocessore.

La maggior parte dei programmi consiste di un prógramma principale e di diverse subroutine. Ciò è vantaggioso poiché si possono usare provate routine ed eseguire il debug e collaudare le altre subroutine separatamente. Bisogna, tuttavia, badare ad usare le subroutine in modo preciso e ricordare i loro esatti effetti sui registri e locazioni di memoria.

DOCUMENTAZIONE DELLA SUBROUTINE

DOCUMENTAZIONE DELLE SUBROUTINE

I listing della subroutine devono fornire sufficienti informazioni in modo che l'utilizzatore non abbia necessità di esaminare la struttura interna della subroutine. Tra le necessarie specificazioni vi sono:

- Una descrizione dello scopo della subroutine.
- Un elenco di parametri di ingresso e di uscita.
- I registri e le locazioni di memoria utilizzati.
- Un caso campione.

Se si seguono queste direttive, la subroutine sarà facile da utilizzare.

ESEMPI

È importante notare che tutti gli esempi seguenti riservano un'area di memoria come Stack RAM. Se il monitor nel vostro microcomputer costituisce una tale area, questa si può utilizzare. Se desiderate cercare di costituire una vostra propria area di Stack, ricordate di conservare e di ripristinare il Puntatore di Stack del monitor onde dar origine ad un ritorno corretto al termine del vostro programma principale.

Per conservare il Puntatore di Stack nel monitor, utilizzare l'istruzione LD (addr), SP. Per ripristinare il Puntatore di Stack del monitor, usare l'istruzione LD SP, (addr). Ambedue queste istruzioni richiedono un codice operativo a due byte (ED 7B per il caricamento del Puntatore di Stack, ED 73 per la sua memorizzazione) oltre ai due byte di indirizzo.

Abbiamo usato l'indirizzo 0080 (esadecimale) come punto di partenza dello Stack. Potreste aver la necessità di sostituire coerentemente quell'indirizzo con uno più adeguato alla vostra configurazione. Dovreste consultare il manuale del vostro microcomputer per determinare le variazioni richieste.

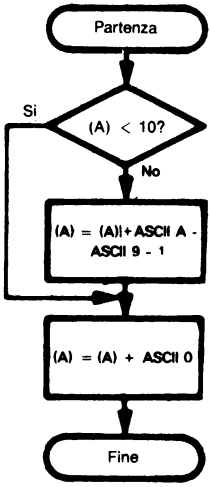
Conversione da Esadecimale ad ASCII

Scopo: Convertire il contenuto dell'Accumulatore in un carattere ASCII. Porre il risultato nell'Accumulatore. Supporre che l'Accumulatore contenga un'unica cifra esadecimale.

Problemi Campione:

a. (A) = 0C
Risultato: (A) = 43 'C'
b. (A) = 06
Risultato: (A) = 36 '6'

Diagramma di Flusso:



Programma Sorgente:

Il programma chiamante inizializza lo Stack alla locazione di memoria 0080, preleva il dato dalla locazione 0040, chiama la subroutine di conversione, e memorizza il risultato nella locazione di memoria 0041.

```
ORG     0
LD      SP,80H     ;FAI PARTIRE LO STACK DALLA LOCAZIONE 0080
LD      A,(40H)    ;PRENDI IL DATO
CALL    ASDEC      ;CONVERTI IL DATO IN ASCII
LD      (41H),A    ;IMMAGAZZINA IL RISULTATO
HALT
```

La subroutine converte una cifra da esadecimale ad ASCII.

```
        ORG     20H
ASDEC:  CP      10          ;IL DATO È UN DIGIT DECIMALE?
        JR      C,ASCZ
        ADD     A,'A'-'9'-1 ;NO, AGGIUNGI L'OFFSET PER LE LETTERE
ASCZ:   ADD     A,'0'       ;CONVERTI IL DATO IN ASCII
        RET
```

Documentazione della subroutine:

```
;
;SUBROUTINE ASDEC
;
;SCOPO: ASDEC CONVERTE UN DIGIT ESADECIMALE NELL'ACCUMULATORE
  IN UN DIGIT ASCII NELL'ACCUMULATORE
;
;CONDIZIONI INIZIALI: DIGIT ESADECIMALE IN A
;
;CONDIZIONI FINALI: CARATTERE ASCII IN A
;
;REGISTRI USATI:A
;
;ESEMPIO CAMPIONE:
;  CONDIZIONI INIZIALI: 6 NELL'ACCUMULATORE
;  CONDIZIONI FINALI: 6 IN ASCII (36 ESADECIMALE)
;     NELL'ACCUMULATORE
;
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)		
1) Programma chiamante				
0000	31		LD	SP,80H
0001	80			
0002	00			
0003	3A		LD	A,(40H)
0004	40			
0005	00			
0006	CD		CALL	ASDEC
0007	20			
0008	00			
0009	32		LD	(41H),A
000A	41			
000B	00			
000C	76		HALT	
2) Subroutine				
0020	FE	ASDEC:	CP	10
0021	0A			
0022	38		JR	C,ASCZ
0023	02			
0024	C6		ADD	A,'A'-'9'-1
0025	07			
0026	C6	ASCZ:	ADD	A,'0'
0027	30			
0028	C9		RET	

L'istruzione LD SP, 80H inizializza lo Stack alla locazione di memoria 0080. Ricordate che lo Stack progredisce verso il basso (verso indirizzi più bassi). Lo Stack si colloca solitamente all'estremità alta della RAM (cioè agli indirizzi più elevati) in modo che non interferisca con altre memorizzazioni temporanee. L'istruzione di Call pone l'indirizzo di partenza della subroutine (0020 esadecimale) nel Contatore del Programma e salva il vecchio Contatore del Programma (0009 esadecimale) all'interno dello Stack. La procedura è:

FASE 1 — Decremento del Puntatore di Stack, salvataggio dei MSB del vecchio Contatore del Programma nello Stack.
FASE 2 — Decremento del Puntatore di Stack, salvataggio dei LSB del vecchio Contatore del Programma nello Stack.

Notate che il Puntatore di Stack dello Z80 contiene sempre l'indirizzo dell'ultima locazione occupata dallo Stack.

Il risultato è:

(007F) = 00
(007E) = 09
(SP) = 007E

Il valore che viene salvato è il valore del Contatore del Programma dopo che il processore ha raccolto l'intera istruzione di Call dalla memoria. Notate che l'indirizzo finisce memorizzato esattamente come gli altri nello Z80, con i bit meno significativi nell'indirizzo più basso.

L'istruzione di Return carica il Contatore del Programma con il contenuto delle due ultime locazioni di memoria appartenenti allo Stack. La procedura è:

FASE 1 — Caricamento di otto bit dallo Stack nei LSB del Contatore del Programma. Incremento del Puntatore di Stack.
FASE 2 — Caricamento di otto bit dallo Stack nei MSB del Contatore del Programma. Incremento del Puntatore di Stack.

Il risultato in tal caso è:

(PC) = (007F) e (007E)
= 0009
(SP) = 0080

Questa subroutine ha un solo parametro d'ingresso e dà luogo a un solo risultato. L'Accumulatore è l'ovvia destinazione di entrambi.

Il programma chiamante implica tre fasi: il collocamento del dato nell'Accumulatore, la chiamata della subroutine, e la memorizzazione del risultato. L'inizializzazione complessiva deve anche collocare lo Stack nell'appropriata area di memoria.

La subroutine è rientrante, dato che non utilizza nessun dato in memoria; è rilocabile, dato che l'indirizzo ASCZ è relativo.

Notate che l'istruzione CALL si risolve nell'esecuzione di quattro o cinque istruzioni le quali richiedono 36 o 38 cicli di clock. La chiamata di una subroutine può richiedere un lungo tempo di esecuzione sebbene si presenti come una sola istruzione nel programma.

Se si pensa di utilizzare lo Stack per i parametri, ricordate che CALL pone l'indirizzo di ritorno in cima allo Stack. Si può incrementare il Puntatore di Stack due volte (INC SP) scavalcare l'indirizzo di ritorno, ma ci si deve anche ricordare di aggiustare correttamente il Puntatore di

Stack prima di eseguire il ritorno. Si può anche trasferire il Puntatore di Stack nei Registri H e L con la sequenza:

```
LD      HL,0
ADD     HL,SP     ;TRASFERISCI IL PUNTATORE DI STACK
                  ;  NEL REGISTRO INDIRIZZI
```

Ora si può utilizzare l'indirizzamento di memoria implicito con H e L per accedere ai dati nello Stack. Un altro metodo alternativo è quello di trasferire il Puntatore di Stack in un registro indice (p. es. IX) con la sequenza:

```
LD      IX,0
ADD     IX,SP     ;TRASFERISCI IL PUNTATORE DI STACK
                  ;  NEL REGISTRO INDIRIZZI
```

Quest'alternativa ha il vantaggio di permettere ora l'accesso a dati ed indirizzi nello Stack con spostamenti indicizzati. Inoltre, la Coppia di Registri HL è immediatamente disponibile per l'uso nella subroutine. Notate che si possono adoperare le istruzioni LD SP, HL oppure LD SP, IX per restituire un valore adatto al Puntatore di Stack.

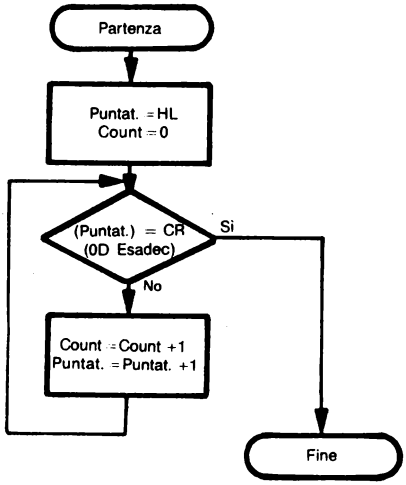
Lunghezza di una Stringa di Caratteri

Scopo: Determinare la lunghezza di una stringa di caratteri ASCII. L'indirizzo di partenza della stringa è nella Coppia di Registri HL. La parte finale della stringa viene contraddistinta da un carattere di ritorno carrello (CR, 0D esadecimale). Porre la lunghezza della stringa (escluso il ritorno carrello) nell'Accumulatore.

Problemi Campione:

```
a.              (HL) = 0043
              (0043) = 0D
   Risultato:    (A) = 00
b.              (HL) = 0043
              (0043) = 52  'R'
              (0044) = 41  'A'
              (0045) = 54  'T'
              (0046) = 48  'H'
              (0047) = 45  'E'
              (0048) = 52  'R'
              (0049) = 0D  CR
   Risultato:    (A) = 06
```

Diagramma di Flusso:



Programma Sorgente:

Il programma chiamante inizializza lo Stack alla locazione di memoria 0080, preleva l'indirizzo di partenza della stringa dalle locazioni di memoria 0040 e 0041, chiama la subroutine per determinare la lunghezza della stringa, e memorizza il risultato nella locazione di memoria 0042.

```
        LD      SP,80H    ;FAI PARTIRE LO STACK DALLA LOCAZIONE 0080
        LD      HL,(40H)  ;PRENDI L'INDIRIZZO DI PARTENZA DELLA STRINGA
        CALL    STLEN     ;DETERMINA LA LUNGHEZZA DELLA STRINGA
        LD      (42H),A   ;MEMORIZZA LA LUNGHEZZA DI STRINGA
        HALT
```

La subroutine determina la lunghezza di una stringa di caratteri ASCII e colloca la lunghezza nell'Accumulatore.

```
        ORG     20H
STLEN:  LD      B,0       ;LUNGHEZZA DI STRINGA = ZERO
        LD      A,0DH     ;PRENDI IL RITORNO CARRELLO IN ASCII
CHKCR:  CP      (HL)      ;IL CARATTERE È UN RITORNO CARRELLO?
        JR      Z,DONE    ;SÌ, FINE DELLA STRINGA
        INC     B         ;NO, SOMMA 1 ALLA LUNGHEZZA DI STRINGA
        INC     HL
        JR      CHKCR
DONE:   LD      A,B
        RET
```

Documentazione della subroutine:

```
;
;SUBROUTINE STLEN
;
;SCOPO: STLEN DETERMINA LA LUNGHEZZA DI UNA STRINGA
;  (NUMERO DI CARATTERI PRIMA DI UN RITORNO CARRELLO)
;
;CONDIZIONI INIZIALI: INDIRIZZO DI PARTENZA DELLA
;  STRINGA NELLA COPPIA DI REGISTRI HL
;
;REGISTRI USATI: A,B,H,L
;
;ESEMPIO CAMPIONE:
;  CONDIZIONI DI PARTENZA: (HL) = 0043
;     (0043) = 35, (0044) = 46, (0045) = 0D
;  CONDIZIONI FINALI: (A) = 02
;
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)		
1) Programma chiamante				
0000	31		LD	SP,80H
0001	80			
0002	00			
0003	2A		LD	HL,(40H)
0004	40			
0005	00			
0006	CD		CALL	STLEN
0007	20			
0008	00			
0009	32		LD	(42H),A
000A	42			
000B	00			
000C	76		HALT	
2) Subroutine				
0020	06	STLEN:	LD	B,0
0021	00			
0022	3E		LD	A,0DH
0023	0D			
0024	BE	CHKCR:	CP	(HL)
0025	28		JR	Z,DONE
0026	04			
0027	04		INC	B
0028	23		INC	HL
0029	18		JR	CHKCR
002A	F9			
002B	78	DONE:	LD	A,B
002C	C9		RET	

Il programma chiamante comporta quattro fasi: inizializzare il Puntatore di Stack, porre l'indirizzo di partenza della stringa nella Coppia di Registri HL, chiamare la subroutine e memorizzare il risultato.

La subroutine è rientrante, dato che non varia il contenuto di nessuna locazione di memoria. Essa è rilocabile, poiché tutte le istruzioni di Salto usano indirizzi relativi.

La subroutine modifica il Registro B e l'indirizzo contenuto nella Coppia di Registri HL come pure l'Accumulatore. Il programmatore deve tener presente che il dato precedentemente immagazzinato nel Registro B e l'indirizzo precedentemente caricato in HL andranno perduti; la documentazione della subroutine deve indicare quali registri vengono utilizzati.

Un ulteriore metodo per non distruggere il contenuto dei registri nella subroutine è quello di salvarli nello Stack e quindi ripristinarli prima del ritorno al programma principale. In questo modo si rende più semplice la routine chiamante, ma ciò porta all'utilizzo di tempo e memoria supplementari (nel programma e nello Stack).

La subroutine ha un solo parametro d'ingresso, il quale è un indirizzo. Il modo migliore per passare questi parametri è attraverso una coppia di registri e, dato che la coppia HL è certamente la più flessibile finché sono interessate opzioni per l'indirizzamento, questa è la scelta più ovvia.

Il sottoprogramma contiene un'istruzione di Salto incondizionato, JR CHKCR. Modificando le condizioni iniziali prima di entrare nel ciclo del sottoprogramma, siete in grado di eliminare questo salto?

Se il carattere finale non fosse sempre un ritorno carrello espresso in codice ASCII, si potrebbe trasformare quel carattere in un altro parametro. Ora il programma chiamante dovrebbe porre il carattere finale nell'Accumulatore e l'indirizzo di partenza della stringa nella Coppia di Registri HL, prima di chiamare la subroutine.

Un modo per passare i parametri che non dipenda da dati variabili è quello di porre i valori nella memoria programmi immediatamente dopo l'istruzione Call. Si può adoperare il vecchio Contatore del Programma (salvato in cima allo Stack) per accedere ai dati, ma si deve regolare correttamente il suo valore prima di restituire il controllo al programma principale. Per esempio, si potrebbe passare il valore del carattere finale tramite questa via. Il programma principale e il sottoprogramma sarebbero i seguenti:

Programma chiamante:

```
        ORG     0
        LD      SP,80H    ;FAI PARTIRE LO STACK DALLA LOCAZIONE 0080
        LD      (HL),40H  ;PRENDI L'INDIRIZZO DI PARTENZA DELLA STRINGA
        CALL    STLEN     ;DETERMINA LA LUNGHEZZA DELLA STRINGA
        DEFB    '.'       ;TERMINATORE = PERIODO IN ASCII
        LD      (42H),A   ;MEMORIZZA LA LUNGHEZZA DI STRINGA
        HALT
```

Subroutine:

```
        ORG     20H
STLEN:  POP     DE        ;PRENDI L'INIZIO DELLA LISTA DEI PARAMETRI
        LD      A,(DE)    ;PRENDI IL CARATTERE DI TERMINAZIONE
        INC     DE        ;AGGIUSTA L'INDIRIZZO DI RITORNO
        PUSH    DE
        LD      B,0       ;LUNGHEZZA DI STRINGA = ZERO
CHKCR:  CP      (HL)      ;È IL CARATTERE DI TERMINAZIONE?
        JR      Z,DONE    ;SÌ, FINE DELLA STRINGA
        INC     B         ;NO, SOMMA 1 ALLA LUNGHEZZA DI STRINGA
        INC     HL
        JR      CHKCR
DONE:   LD      A,B
        RET
```

Questa subroutine è più lunga ed utilizza la Coppia di Registri DE, ma il programma chiamante non ha necessità di caricare il carattere finale in un registro. L'istruzione INC DE è necessaria per forzare un ritorno alla successiva istruzione, invece che alla lista dei parametri[1].

POP e PUSH trasferiscono il contenuto di coppie di registri o registri indice da e verso lo Stack RAM. Gli otto bit meno significativi sono trasferiti per primi e memorizzati per ultimi per conservare una coerenza con il metodo a ritroso di memorizzazione di indirizzi a 16 bit relativo allo Z80. Ricordate che lo Stack RAM progredisce verso il basso (verso gli indirizzi più bassi).

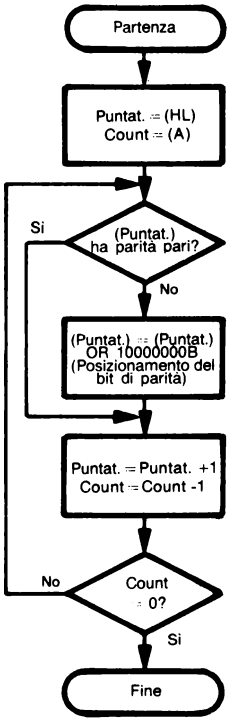
Aggiunta della Parità di Tipo Pari a Caratteri ASCII

Scopo: Aggiungere la parità di tipo pari ad una stringa di caratteri ASCII a 7 bit. La lunghezza della stringa è contenuta nell'Accumulatore e l'indirizzo di partenza della stringa è nella Coppia di Registri HL. Porre la parità di tipo pari nel bit più significativo di ciascun carattere, cioè posizionare il bit più significativo ad 1 se ciò rende pari il numero totale di bit 1 nella parola.

Problemi Campione:

	(A)	=	06
	(HL)	=	0041
	(0041)	=	31
	(0042)	=	32
	(0043)	=	33
	(0044)	=	34
	(0045)	=	35
	(0046)	=	36
Risultato:	(0041)	=	B1
	(0042)	=	B2
	(0043)	=	33
	(0044)	=	B4
	(0045)	=	35
	(0046)	=	36

Diagramma di Flusso:



Programma Sorgente:

Il programma chiamante inizializza lo Stack alla cella di memoria 0080, dispone l'indirizzo di partenza della stringa a 0041, preleva la lunghezza della stringa dalla locazione di memoria 0030, e chiama la subroutine della parità di tipo pari.

```
        ORG     0
        LD      SP,80H    ;FAI PARTIRE LO STACK DALLA LOCAZIONE 0080
        LD      HL,41H    ;PRENDI L'INDIRIZZO DI PARTENZA DELLA STRINGA
        LD      A,(30H)   ;PRENDI LA LUNGHEZZA DI STRINGA
        CALL    EPAR
        HALT
```

La subroutine aggiunge la parità di tipo pari ad una stringa di caratteri ASCII.

```
        ORG     20H
EPAR:   LD      B,A
        LD      C,10000000B   ;PRENDI IL BIT DI PARITÀ 1
SETPR:  LD      A,(HL)        ;PRENDI IL CARATTERE
        OR      C             ;POSIZIONA A 1 IL BIT DI PARITÀ
        JP      P0,CHCNT      ;LA PARITÀ ORA È PARI?
        LD      (HL),A        ;SÌ, SALVA IL CARÁTTERE CON PARITÀ PARI
CHCNT:  INC     HL
        DJNZ    SETPR
        HALT
```

Documentazione della subroutine:

```
;
;SUBROUTINE EPAR
;
;SCOPO: EPAR AGGIUNGE UNA PARITÀ PARI AD UNA STRINGA DI
;   CARATTERI ASCII A 7 BIT
;
;CONDIZIONI INIZIALI: INDIRIZZO DI PARTENZA DELLA STRINGA IN
;   HL, LUNGHEZZA DELLA STRINGA IN A
;
;CONDIZIONI FINALI: PARITÀ PARI NEL BIT MSB DI OGNI CARATTERE
;
;REGISTRI USATI: A, B, C, H, L.
;
;ESEMPIO CAMPIONE:
;   CONDIZIONI INIZIALI: (HL) = 0041
;   (A) = 2, (0041) = 32, (0042) = 33
;   CONDIZIONI FINALI: (0041) = B2, (0042) = 33
;
```

Questa subroutine possiede due parametri, un indirizzo ed un numero. La Coppia di Registri HL è utilizzata per passare l'indirizzo e l'Accumulatore per passare il numero. Non vengono prodotti dei risultati espliciti, dato che la subroutine ha influenza solamente sui MSB di ciascun carattere compreso nella stringa.

Il programma chiamante deve porre l'indirizzo di partenza della stringa nella Coppia di Registri HL e la lunghezza della stringa nell'Accumulatore prima di trasferire il controllo alla subroutine. La subroutine modifica i valori contenuti nei Registri A, H e L e adopera i Registri B e C come memorizzazione temporanea. Esso è rientrante, dato che non usa nessuna locazione di memoria fissa come memorizzazione temporanea.

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)		
1) Programma chiamante				
0000	31		LD	SP,80H
0001	80			
0002	00			
0003	21		LD	HL,41H
0004	41			
0005	00			
0006	3A		LD	A,(30H)
0007	30			
0008	00			
0009	CD		CALL	EPAR
000A	20			
000B	00			
000C	76		HALT	
2) Subroutine				
0020	47	EPAR:	LD	B,A
0021	0E		LD	C,10000000B
0022	80			
0023	7E	SETPR:	LD	A,(HL)
0024	B1		OR	C
0025	E2		JP	PO,CHCNT
0026	29			
0027	00			
0028	77		LD	(HL),A
0029	23	CHCNT:	INC	HL
002A	10		DJNZ	SETPR
002B	F7			
002C	C9		RET	

Confronto con Campione

Scopo: Confrontare due stringhe di caratteri ASCII per vedere se sono gli stessi. La lunghezza delle stringhe è contenuta nell'Accumulatore. L'indirizzo di partenza di una stringa è contenuta nella Coppia di Registri HL; l'indirizzo di partenza dell'altra è nella Coppia di Registri DE. Se le due stringhe sono identiche, azzerare l'Accumulatore; altrimenti, posizionare l'Accumulatore ad FF (esadecimale).

Problemi Campione:

a.

(A)	=	03	
(DE)	=	50	
(HL)	=	60	
(0050)	=	43	'C'
(0051).	=	41	'A'
(0052)	=	54	'T'
(0060)	=	43	'C'
(0061)	=	41	'A'
(0062)	=	54	'T'

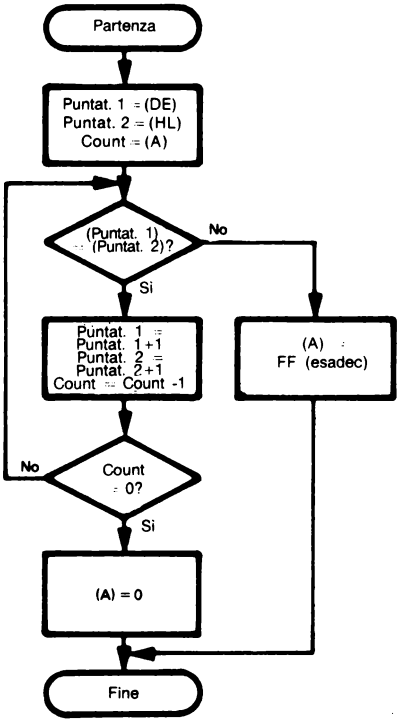
Risultato: (A) = 0, poiché le stringhe sono le stesse.

b.

(A)	=	03	
DE	=	50	
(HL)	=	60	
(0050)	=	52	'R'
(0051)	=	41	'A'
(0052)	=	54	'T'
(0060)	=	43	'C'
(0061)	=	41	'A'
(0062)	=	54	'T'

Risultato: (A) = FF (esadec), poiché i primi caratteri sono diversi.

Diagramma di Flusso:



Programma Sorgente:

Il programma chiamante inizializza lo Stack alla locazione di memoria 0080, dispone gli indirizzi di partenza delle stringhe a 0050 e 0060, rispettivamente, preleva la lunghezza della stringa dalla locazione di memoria 0040, chiama la subroutine di confronto con il campione, e pone il risultato nella locazione di memoria 0041.

```
        ORG     0
        LD      SP,80H    ;FAI PARTIRE LO STACK DALLA LOCAZIONE 0080
        LD      DE,60H    ;PRENDI L'INDIRIZZO DI PARTENZA DELLA STRINGA 1
        LD      HL,50H    ;PRENDI L'INDIRIZZO DI PARTENZA DELLA STRINGA 2
        LD      A,(40H)   ;PRENDI LA LUNGHEZZA DI STRINGA
        CALL    PMTCH     ;CONTROLLA SE SONO UGUALI
        LD      (41H),A   ;SALVA L'INDICATORE DI UGUAGLIANZA
        HALT
```

La subroutine determina se le due stringhe sono le stesse.

```
        ORG     20H
PMTCH:  LD      B,A       ;COUNT = LUNGHEZZA DI STRINGA
        LD      C,0FFH    ;INDICATORE = FF (ESADEC)
                          ;   PER NESSUNA UGUAGLIANZA
CHCAR:  LD      A,(DE)    ;PRENDI IL CARATTERE DALLA STRINGA 1
        CP      (HL)      ;C'È UGUAGLIANZA CON LA STRINGA 2?
        JR      NZ,DONE ;NO, VAI A DONE — LE STRINGHE NON COINCIDONO
        INC     DE
        INC     HL
        DJNZ    CHCAR
        LD      C,0       ;INDICE = ZERO, LE STRINGHE COINCIDONO
DONE:   LD      A,C
        RET
```

Documentazione della subroutine:

```
;
;SUBROUTINE PMTCH
;
;SCOPO: PMTCH DETERMINA SE DUE STRINGHE
;  SONO EQUIVALENTI
;
;CONDIZIONI INIZIALI: INDIRIZZI DI PARTENZA DELLE
;  STRINGHE IN DE ED HL,
;     LUNGHEZZE DELLE STRINGHE NELL'ACCUMULATORE
;
;CONDIZIONI FINALI: 0 IN A SE LE STRINGHE SONO
;  UGUALI, FF IN A SE DIVERSE
;
;REGISTRI USATI: A, B, D, E, H, L.
;
ESEMPIO CAMPIONE:
;  CONDIZIONI DI PARTENZA: (HL) = 0050
;     (DE) = 0060, (A) = 2
;     (0050) = 36, (0051) = 39
;     (0060) = 36, (0061) = 39
;     CONDIZIONI FINALI: (A) = 0 POICHÈ LE STRINGHE SONO UGUALI
;
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)		
1) Programma chiamante				
0000	31		LD	SP,80H
0001	80			
0002	00			
0003	11		LD	DE,60H
0004	60			
0005	00			
0006	21		LD	HL,50H
0007	50			
0008	00			
0009	3A		LD	A,(40H)
000A	40			
000B	00			
000C	CD		CALL	PMTCH
000D	20			
000E	00			
000F	32		LD	(41H),A
0010	41			
0011	00			
0012	76		HALT	
2) Subroutine				
0020	47	PMTCH:	LD	B,A
0021	0E		LD	C,0FFH
0022	FF			
0023	1A	CHCAR:	LD	A,(DE)
0024	BE		CP	(HL)
0025	20		JR	NZ,DONE
0026	06			
0027	13		INC	DE
0028	23		INC	HL
0029	10		DJNZ	CHCAR
002A	F8			
002B	0E		LD	C,0
002C	00			
002D	79	DONE:	LD	A,C
002E	C9		RET	

Questa subroutine, come le precedenti, cambia tutti i flag. Si dovrebbe generalmente supporre che una subroutine modifichi i flag salvo avviso specificatamente contrario. Se il programma principale necessita dei vecchi valori dei flag (per verifiche successive), esso deve salvarli nello Stack prima di chiamare la subroutine. Ciò viene eseguito mediante l'istruzione PUSH AF.

La subroutine è rientrante e fa variare i contenuti di tutti i registri principali tranne C.

Questa subroutine possiede tre parametri — i due indirizzi di partenza e la lunghezza delle stringhe. Questi parametri si servono di cinque registri di scopo generale.

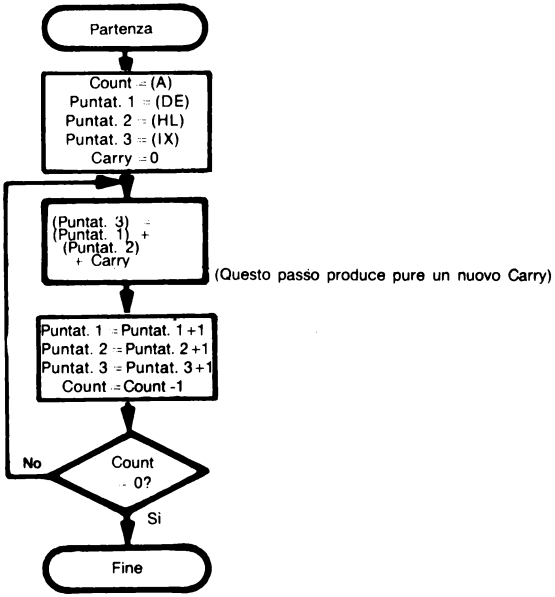
Addizione in Multi-precisione

Scopo: Sommare due numeri binari multi-byte. La lunghezza dei numeri in byte è contenuta nell'Accumulatore. Gli indirizzi di partenza dei numeri sono nelle Coppie di Registri HL e DE. L'indirizzo di partenza del risultato è contenuto nel Registro Indice IX. Tutti i numeri hanno inizio con i bit meno significativi.

Problema Campione:

```
      (A) = 04
     (DE) = 51
     (HL) = 61
     (IX) = 71
   (0051) = C3
   (0052) = A7
   (0053) = 5B
   (0054) = 2F
   (0061) = B8
   (0062) = 35
   (0063) = DF
   (0064) = 14
Risultato: = (0071) = 7B
             (0072) = DD
             (0073) = 3A
             (0074) = 44
     cioè    2F5BA7C3
           + 14DF35B8
             --------
             443ADD7B
```

Diagramma di Flusso:



Programma Sorgente:

Il programma chiamante inizializza lo Stack dalla locazione di memoria 0080, dispone gli indirizzi di partenza dei vari numeri a 0050, 0060 e 0070, rispettivamente, preleva la lunghezza dei numeri dalla locazione di memoria 0040, e chiama la subroutine relativa alla addizione multiprecisione.

```
        ORG     0
        LD      SP,80H   ;FAI PARTIRE LO STACK ALLA LOCAZIONE 0080
        LD      HL,50H   ;PRENDI L'INDIRIZZO DI PARTENZA DELLA STRINGA 1
        LD      DE,60H   ;PRENDI L'INDIRIZZO DI PARTENZA
                         ;   DEL SECONDO NUMERO
        LD      IX,70H   ;PRENDI L'INDIRIZZO DI PARTENZA DEL RISULTATO
        LD      A,(40H)  ;PRENDI LA LUNGHEZZA DEI NUMERI IN BYTE
        CALL    MPADD    ;ADDIZIONE IN PRECISIONE MULTIPLA
        HALT
```

La subroutine esegue l'addizione binaria a multi-precisione.

```
        ORG     20H
MPADD:  LD      B,A      ;COUNT = LUNGHEZZA DEI NUMERI IN BYTE
        AND     A        ;AZZERA IL CARRY PER L'AVVIO
ADDW:   LD      A,(DE)   ;PRENDI LA PAROLA DAL PRIMO NUMERO
        ADC     A,(HL)   ;AGGIUNGI UNA PAROLA DAL SECONDO NUMERO
        LD      (IX),A   ;IMMAGAZZINA UNA PAROLA DEL RISULTATO
        INC     DE
        INC     HL
        INC     IX
        DJNZ    ADDW
        RET
```

Documentazione della subroutine:

```
;
;SUBROUTINE MPADD
;
;SCOPO: MPADD SOMMA DUE NUMERI BINARI A PIÙ BYTE
;
;CONDIZIONI INIZIALI: INDIRIZZI DI PARTENZA DEI NUMERI IN D ED E,
;   H ED L; INDIRIZZO DI PARTENZA DEL RISULTATO IN IX, LUNGHEZZA
;   DEI NUMERI IN A.
;
;REGISTRI USATI: A, B, D, E, H, L, IX.
;
;ESEMPIO CAMPIONE:
;   CONDIZIONI DI PARTENZA: (HL) = 0050, (DE) = 0060, (IX) = 0070,
      (A) = 2, (0050) = C3, (0051) = A7, (0060) = B8, (0061) = 35
;CONDIZIONI FINALI: (0070) = 7B, (0071) = DD
;
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)	
1) Programma chiamante			
0000	31	LD	SP,80H
0001	80		
0002	00		
0003	21	LD	HL,50H
0004	50		
0005	00		
0006	11	LD	DE,60H
0007	60		
0008	00		
0009	DD	LD	IX,70H
000A	21		
000B	70		
000C	00		
000D	3A	LD	A,(40H)
000E	40		
000F	00		
0010	CD	CALL	MPADD
0011	20		
0012	00		
0013	76	HALT	
2) Subroutine			
0020	47	LD	B,A
0021	A7	AND	A
0022	1A	LD	A,(DE)
0023	8E	ADC	A,(HL)
0024	DD	LD	(IX),A
0025	77		
0026	00		
0027	13	INC	DE
0028	23	INC	HL
0029	DD	INC	IX
002A	23		
002B	10	DJNZ	ADDW
002C	F5		
002D	C9	RET	

Si utilizza il Registro Indice per contenere l'indirizzo del risultato. Provate a modificare il programma onde usare la Coppia di Registri BC per questo scopo. Cosa accade al contatore?

Si potrebbe anche porre l'indirizzo del risultato in cima allo Stack. L'istruzione EX (SP), HL cambia la cima dello Stack con la Coppia di Registri HL. Variate il programma in modo da utilizzare questa istruzione, ma ricordate di incrementare tutti i tre puntatori dopo ogni ripetizione.

Questa subroutine ha quattro parametri – tre indirizzi e la lunghezza dei numeri. Si usano 6 registri ad 8 bit e il Registro Indice IX a 16 bit per il passaggio dei parametri.

PROBLEMI

Notate che si deve scrivere sia un programma chiamante per il problema campione che una subroutine correttamente documentata.

1) Da ASCII ad Esadecimale

Scopo: Convertire il contenuto dell'Accumulatore dalla rappresentazione di tipo ASCII di una cifra esadecimale alla rappresentazione binaria a 4 bit della cifra. Porre il risultato nell'Accumulatore.

Problemi Campione:

a. (A) = 43 'C'
Risultato: (A) = 0C
b. (A) = 36 '6'
Risultato: (A) = 06

2) Lunghezza di un Messaggio ASCII

Scopo: Determinare la lunghezza di un messaggio codificato in ASCII. L'indirizzo di partenza della stringa di caratteri nel quale è situato il messaggio è contenuto nella Coppia di Registri HL. Il messaggio stesso inizia con un carattere ASCII STX (02 esadecimale) e termina con un altro carattere ASCII ETX (03 esadecimale). Porre la lunghezza del messaggio (il numero di caratteri compresi tra STX e ETX) nell'Accumulatore.

Problema Campione:

(HL) = 0041
(0041) = 49
(0042) = 02 STX
(0043) = 47 'G'
(0044) = 4F 'O'
(0045) = 03 ETX
Risultato: (A) = 02

3) Controllo della Parità di Tipo Pari in Caratteri ASCII

Scopo: Controllare la parità di tipo pari di una stringa di caratteri ASCII. La lunghezza della stringa è contenuta nell'Accumulatore e l'indirizzo di partenza della stessa è invece nella Coppia di Registri HL. Se la parità di tutti i caratteri della stringa è corretta, azzerare l'Accumulatore; altrimenti posizionare quest'ultimo al valore FF esadecimale (tutti i bit ad uno).

Problemi Campione:

a. (A) = 03
(HL) = 0042
(0042) = B1
(0043) = B2
(0044) = 33
Risultato: (A) = 00, dato che tutti i caratteri hanno una parità di tipo pari
b. (A) = 03
(HL) = 0042
(0042) = B1
(0043) = B6
(0044) = 33
Risultato: (A) = FF, dato che il carattere nella locazione di memoria 0043 non possiede una parità di tipo pari.

4) Confronto tra Stringhe

Scopo: Confrontare due stringhe di caratteri ASCII per vedere quale è la maggiore (cioè, quale dovrebbe seguire l'altra in ordine alfabetico).

La lunghezza delle stringhe è contenuta nell'Accumulatore; l'indirizzo di partenza della stringa 1 è nella Coppia di Registri HL, e l'indirizzo di partenza della stringa 2 è nella Coppia di Registri DE. Se la stringa 1 è maggiore od eguale alla stringa 2, azzerare l'Accumulatore; altrimenti posizionare quest'ultimo al valore FF esadecimale (tutti i bit ad uno).

Problemi Campione:

a.
(A) = 03
(DE) = 0060
(HL) = 0050
(0050) = 43 'C'
(0051) = 41 'A'
(0052) = 54 'T'
(0062) = 42 'B'
(0063) = 41 'A'
(0064) = 54 'T'
Risultato = (A) = 00, poiché CAT è maggiore di BAT

b.
(A) = 03
(DE) = 0060
(HL) = 0050
(0050) = 44 'D'
(0051) = 4F 'O'
(0052) = 47 'G'
(0060) = 44 'D'
(0061) = 4F 'O'
(0062) = 47 'G'
Risultato = (A) = 00, poiché le due stringhe sono uguali

c.
(A) = 03
(DE) = 0060
(HL) = 0050
(0050) = 43 'C'
(0051) = 41 'A'
(0052) = 54 'T'
(0060) = 43 'C'
(0061) = 55 'U'
(0062) = 54 'T'
Risultato = (A) = FF (esadec), poiché CUT è maggiore di CAT

5) Sottrazione Decimale

Scopo: Sottrarre un numero decimale (BCD) composto da più di una cifra da un altro. La lunghezza dei numeri (in byte) è contenuta nell'Accumulatore e gli indirizzi di partenza dei numeri sono nelle Coppie di Registri DE ed HL. Sottrarre il numero che ha l'indirizzo di partenza in HL da quello che ha l'indirizzo di partenza in DE. L'indirizzo di partenza del risultato è nel Registro Indice IX. Tutti i numeri iniziano con le cifre meno significative. Il segno del risultato viene restituito all'Accumulatore – zero se il risultato è positivo, FF (esadecimale) se invece è negativo.

Problema Campione:

	(A)	=	04
	(DE)	=	0050
	(HL)	=	0060
	(IX)	=	0070
	(0050)	=	85
	(0051)	=	19
	(0052)	=	70
	(0053)	=	36
	(0060)	=	59
	(0061)	=	34
	(0062)	=	66
	(0063)	=	12
Risultato:	(A)	=	00 (positivo)
	(0070)	=	26
	(0071)	=	85
	(0072)	=	03
	(0073)	=	24

cioè

```
 36701985
-12663459
---------
+24038526
```

BIBLIOGRAFIA

1) Vi sono altri esempi di questa tecnica (per il microprocessore 8080) in S. Mazor e C. Pitchford, "Develop Cooperative Microprocessor Subroutines", Electronic Design, 7 Giugno 1978, pagg. 116-118.

Capitolo 11
INGRESSO/USCITA

Esistono due problemi nel progetto di sezioni di ingresso/uscita: uno è come interfacciare periferiche al computer e trasferire segnali di dati, di stato e di controllo; l'altro è come indirizzare dispositivi di I/O in modo tale che la CPU possa selezionare uno specifico per un trasferimento di dati. Chiaramente il primo problema è più complesso e più interessante dell'altro. Tratteremo perciò qui l'interfacciamento di periferiche e lasceremo il problema dell'indirizzamento ad un libro maggiormente orientato all'hardware.

I/O E MEMORIA

In teoria, il trasferimento di dati verso o da un dispositivo di I/O è simile al trasferimento di dati verso o da la memoria. Infatti, possiamo considerare la memoria proprio come un altro dispositivo di I/O. Comunque, la memoria è speciale per i seguenti motivi:

1) Essa funziona quasi alla stessa velocità del processore.
2) Essa usa lo stesso tipo di segnali della CPU. I soli circuiti usualmente necessari per interfacciare la memoria alla CPU sono driver, ricevitori e traslatori di livello.
3) Essa non richiede formati speciali né segnali di controllo tranne un impulso di Lettura/Scrittura.
4) Essa incamera automaticamente i dati che le sono spediti.
5) La lunghezza della sua parola è la stessa di quella del computer.

La maggior parte dei dispositivi di I/O non hanno tali convenienti caratteristiche. Essi possono funzionare a velocità molto più lente del processore: per esempio, una telescrivente può trasferire soltanto 10 caratteri al secondo, mentre un processore lento può trasferire 10.000 caratteri al secondo. Il campo di velocità è anche molto ampio. Certi sensori possono fornire una lettura al minuto, mentre video display o floppy disk possono trasferire 250.000 bit al secondo. Inoltre, i dispositivi di I/O possono richiedere segnali continui (motori o termometri), correnti piuttosto che tensioni (telescriventi), o tensioni a livelli molto differenti dai segnali impiegati dal processore (display a scarica di gas). I dispositivi di I/O possono anche richiedere speciali formati, procotolli, o segnali di controllo. Le loro lunghezze di parola possono essere molto più corte o molto più lunghe della lunghezza di parola del computer. Queste differenze rendono difficile il progetto di sezioni di I/O e fanno sì che ogni periferica presenti un suo speciale problema di interfacciamento.

CATEGORIE DI I/O

Possiamo, dunque, fornire una descrizione generale di dispositivi e di metodi di interfacciamento. Possiamo grossolanamente separare i dispositivi in tre classi, basate sulle velocità dei loro dati:

1) Dispositivi lenti che cambiano stato non più di una volta al secondo. Il mutamento del loro stato richiede tipicamente dei millisecondi o tempi più lunghi. Tali dispositivi comprendono display luminosi, interruttori (switch), relé, e molti sensori ed otturatori meccanici.

2) Dispositivi a media velocità che trasferiscono dati a velocità da 1 a 10.000 bit al secondo. Tali dispositivi comprendono tastiere, stampanti, lettori di schede, lettori e perforatori di nastro, cassette, linee ordinarie di comunicazione ed un gran numero di sistemi di acquisizione di dati analogici.
2) Dispositivi ad alta velocità che trasferiscono dati a velocità superiori a 10.000 bit al secondo. Tali dispositivi comprendono nastri magnetici, dischi magnetici, stampanti di linea ad alta velocità, linee di comunicazione ad alta velocità e video display.

INTERFACCIAMENTO CON DISPOSITIVI LENTI

L'interfacciamento di dispositivi lenti è semplice. Sono necessari pochi segnali di controllo a meno che i dispositivi non siano multiplexed, cioè parecchi sono gestiti da una sola porta come mostrato nelle Figure da 11-1 a 11-4. I dati d'ingresso da dispositivi lenti non necessitano di essere messi in registri, poiché esso rimane stabile per un lungo intervallo di tempo. I dati di uscita devono, naturalmente, essere messi in registri. I soli problemi in ingresso sono costituiti dalle transizioni che si presentano mentre il computer sta leggendo il dato. Monostabili, latch di tipo «cross-coupled», o programmi di ritardo in software possono limare le transizioni.

Una singola porta può gestire parecchi dispositivi lenti. La Figura 11-1 mostra un demultiplexer che automaticamente dirige il successivo dato di uscita verso il prossimo dispositivo contando le operazioni di uscita. La Figura 11-2 mostra una porta di controllo che fornisce gli ingressi di selezione ad un demultiplexer. Le uscite di dati possono servire qui in un ordine qualunque, ma per cambiare lo stato della porta di controllo è necessaria una ulteriore istruzione di uscita. Le uscite dei demultiplexer sono comunemente impiegate per pilotare parecchi display dalla stessa porta di uscita. Le Figure 11-3 e 11-4 mostrano le stesse alternative per un multiplexer di ingresso.

Si noti la differenza tra ingresso ed uscita con dispositivi lenti:

1) Il dato di ingresso non deve essere messo in un registro, poiché il dispositivo di ingresso mantiene il dato per un intervallo di tempo enorme rispetto ai tempi standard del computer. I dati di uscita devono essere messi su registri, poiché il dispositivo di uscita non risponderà a dati che si presentino solo per alcuni cicli di clock della CPU.
2) Le transizioni di ingresso provocano problemi a causa della loro durata, brevi transizioni di uscita provocano problemi perché i dispositivi di uscita (o gli osservatori) reagiscono lentamente.
3) Le maggiori limitazioni sull'ingresso sono il tempo di reazione e il tempo di risposta, le maggiori limitazioni sull'uscita sono il tempo di risposta e l'osservabilità.

INTERFACCIAMENTO CON DISPOSITIVI A MEDIA VELOCITÀ

I dispositivi a media velocità devono essere sincronizzati in qualche modo al clock del processore. La CPU non può trattare questi dispositivi semplicemente come se mantenessero i loro dati per sempre o come se potessero ricevere dati in ogni istante di tempo. Invece, la CPU deve essere in grado di determinare quando un dispositivo ha nuovi dati di ingresso o è pronta a ricevere dati di uscita. Esso deve anche avere un modo di informare un dispositivo che un nuovo dato di uscita è disponibile o che il precedente dato di ingresso è stato accettato. Si noti che la periferica può essere o contenere un altro processore.

HANDSHAKE

La procedura standard di tipo «unloked» è l'«handshake». Qui il mittente indica al ricevitore la disponibilità di un dato e trasferisce il dato: il ricevitore completa l'«handshake» riconoscendo la ricezione del dato. Il ricevitore può controllare la situazione richiedendo inizialmente il dato o indicando la sua disponibilità ad accettare dati; il mittente allora spedisce il dato e completa l'«handshake» indicando che il dato è disponibile. In un caso o nell'altro, il mittente riconosce che il trasferimento è stato completato con successo e il ricevitore riconosce quando il nuovo dato è disponibile.

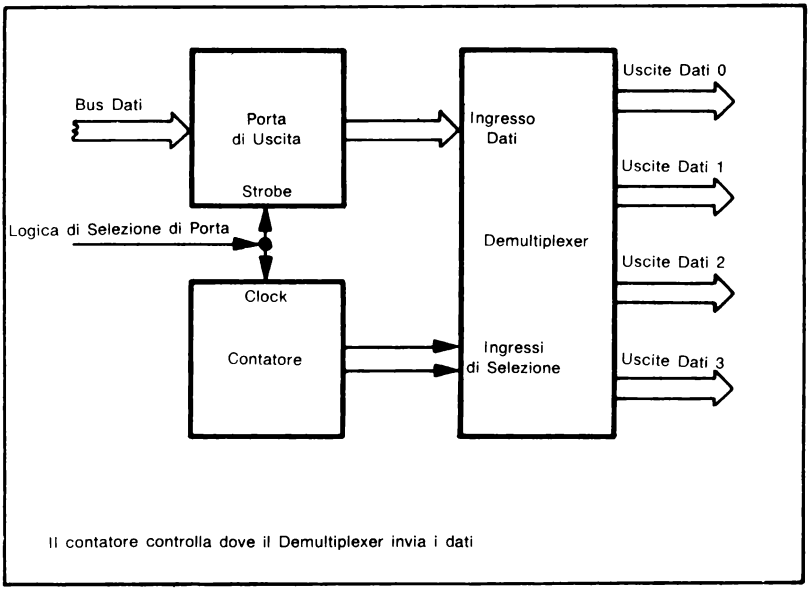


Figura 11-1. Un Demultiplexer di Uscita Controllato da un Contatore

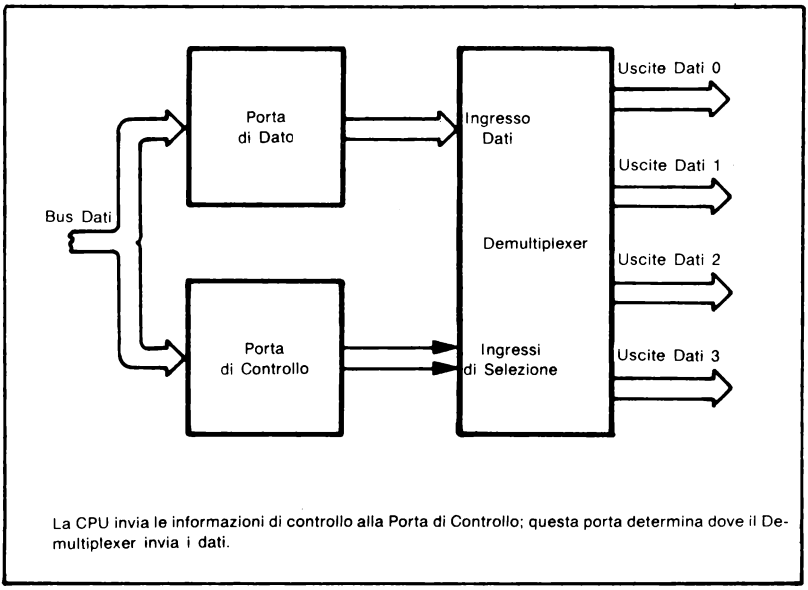


Figura 11-2. Un Demultiplexer di Uscita Controllato da una Porta

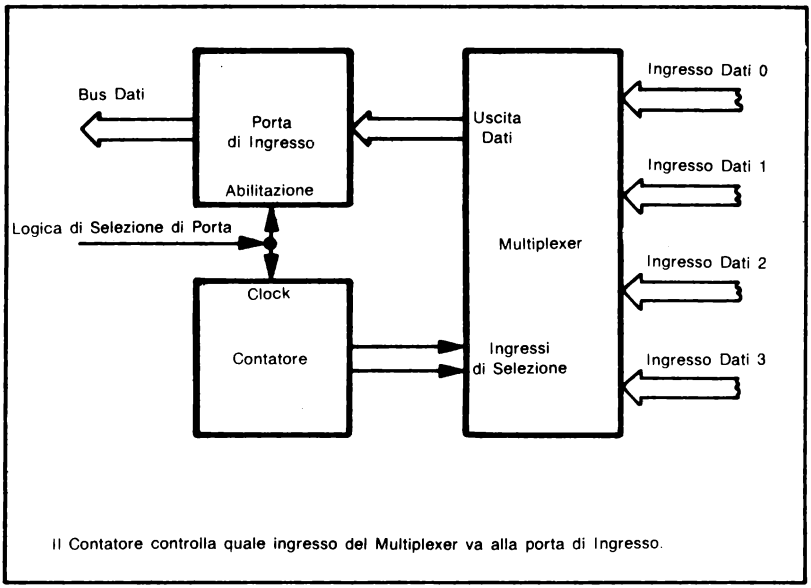


Figura 11-3. Un Multiplexer Controllato da un Contatore

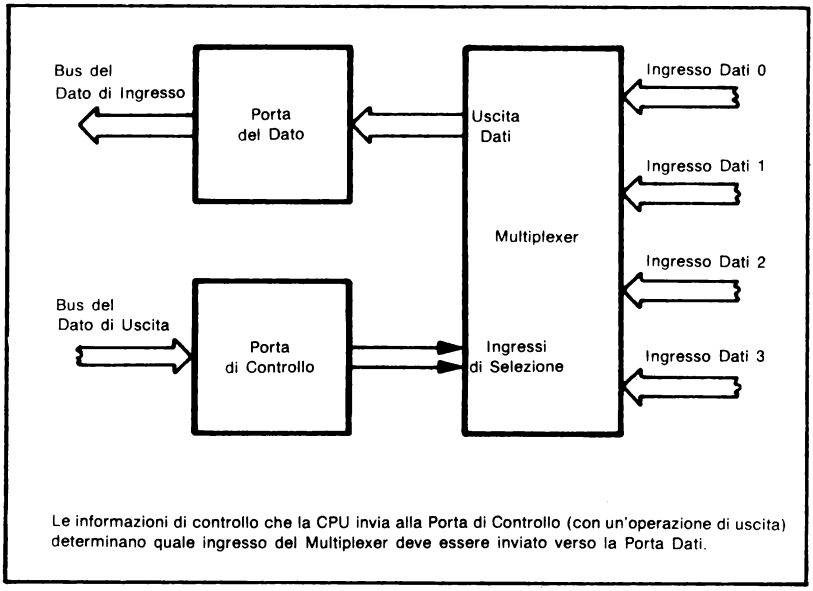


Figura 11-4. Un Multiplexer di Ingresso Controllato da una Porta

Le Figure 11-5 e 11-6 mostrano tipiche operazioni d'ingresso e d'uscita utilizzanti il metodo «handshake». La procedura secondo la quale la CPU controlla l'indisponibilità della periferica prima del trasferimento del dato è detta «polling». Chiaramente, il polling può occupare larga parte del tempo del processore se vi sono molti dispositivi di I/O. Esistono parecchi modi di fornire segnali di «handshake». Tra questi si hanno:

- Separate linee di I/O dedicate. Il processore può trattare linee come porte di I/O aggiuntive o mediante linee speciali o interrupt. Il processore Z80 non ha linee di I/O seriali, contrariamente al dispositivo di Ingresso/Uscita Parallelo Z80 (o PIO).
- Speciali configurazioni sulle linee di I/O. Queste possono essere o singoli bit di start e di stop o interi caratteri o gruppi di caratteri. Le configurazioni devono essere facilmente distinguibili dal rumore di fondo o da stati inattivi.

STROBE

Spesso chiamiamo con la parola «strobe» una linea di I/O separata che indica la disponibilità di un dato o l'evento di un trasferimento. Uno «strobe» può per esempio, mettere un dato in un registro o andare a prendere un dato da un buffer.

Molte periferiche trasferiscono dati ad intervalli regolari: cioè in modo sincrono. In questo caso il solo problema è di far iniziare il processo allineandolo al primo ingresso o contrassegnando la prima uscita. In qualche caso, la periferica fornisce un ingresso di clock dal quale il processore può ottenere informazioni per temporizzazioni.

RIDUZIONE DEGLI ERRORI DI TRASMISSIONE

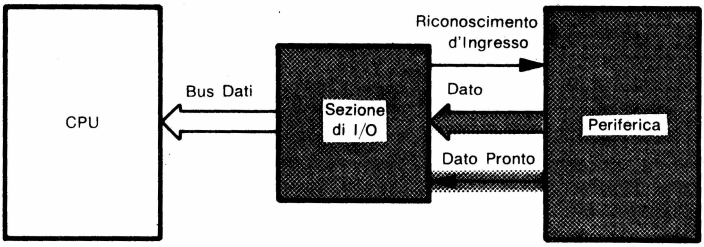
I dispositivi a media velocità hanno problemi di errori di trasmissione. Parecchi metodi possono ridurre la probabilità di tali errori; essi comprendono:

- Il campionamento dei dati di ingresso al centro dell'intervallo di trasmissione in modo da evitare gli effetti dei fronti; cioè tenersi lontano dai fronti dove il dato stia cambiando.
- Il campionamento di ogni ingresso parecchie volte e l'impiego di una logica maggioritaria, quale «tre volte su cinque».[1]
- La generazione ed il controllo della parità: si usa un altro bit tale da rendere pari o dispari il numero dei bit ad «1» nel dato corretto.
- L'impiego di codici a rivelazione o correzione d'errore come «checksum» LRC (controllo di ridondanza longitudinale), e CRC (controllo di ridondanza ciclica).[2]

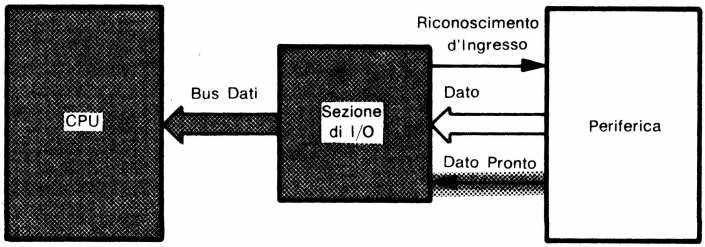
INTERFACCIAMENTO CON DISPOSITIVI AD ALTA VELOCITÀ

ACCESSO DIRETTO IN MEMORIA

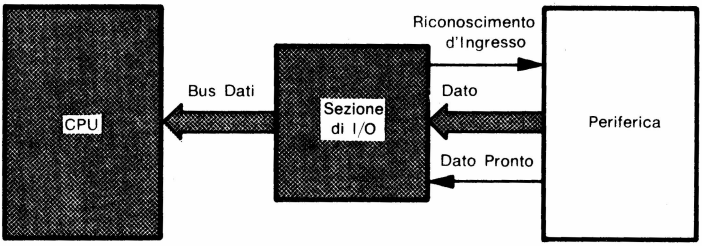
I dispositivi ad alta velocità che trasferiscono più di 10.000 bit al secondo richiedono metodi speciali. La tecnica usuale spinge a costruire una speciale funzione di controllo che trasferisce direttamente i dati tra la memoria e il dispositivo di I/O. Questo processo è chiamato accesso diretto in memoria (DMA). Il controllore DMA deve togliere la CPU dai bus, fornire gli indirizzi ed i segnali di controllo alla memoria e trasferire i dati. Tale controllore sarà abbastanza complesso, essendo tipicamente realizzato con 50÷100 chip, quantunque siano ora disponibili strutture LSI.[3] La CPU deve inizialmente caricare i contatori di Indirizzo e di Dato nel controllore in modo tale che il controllore saprà da dove partire e quanto trasferire.



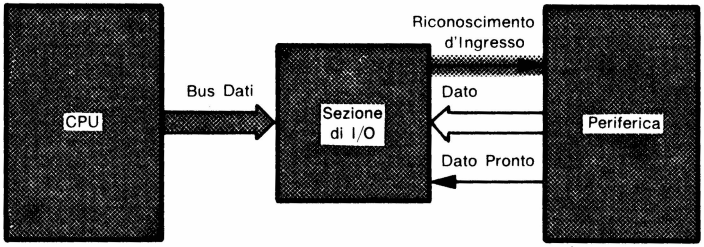
a) La Periferica fornisce il dato e il segnale Dato Pronto alla sezione di I/O del computer.



b) La CPU legge il segnale Dato Pronto dalla sezione di I/O (questo può essere un collegamento d'interrupt in hardware).

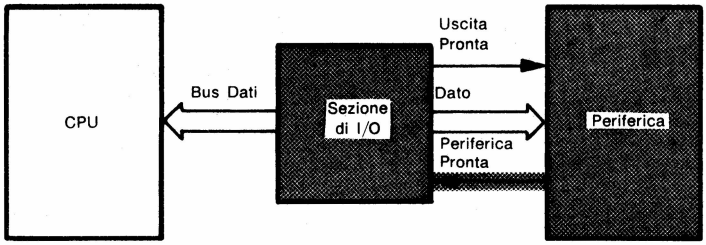


c) La CPU legge il dato dalla sezione di I/O.

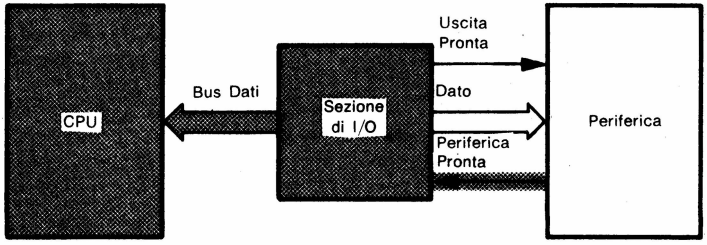


d) La CPU invia il segnale Riconoscimento di Ingresso alla sezione di I/O, che a sua volta fornisce il segnale Riconoscimento di Ingresso alla Periferica (questo può essere un collegamento hardware).

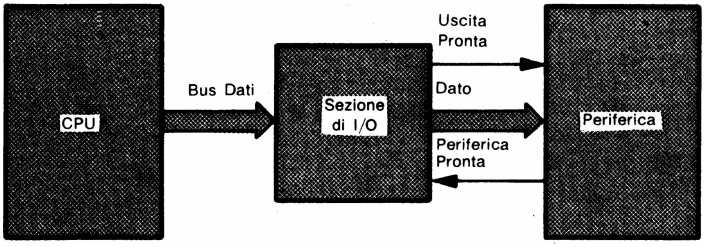
Figura 11-5. Un Handshake di Ingresso.



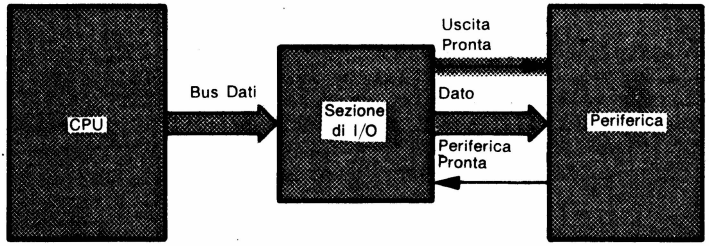
a) La Periferica fornisce il segnale Periferica Pronta alla sezione di I/O del computer.



b) La CPU legge il segnale Periferica Pronta dalla sezione di I/O (questo può essere un collegamento d'interrupt hardware).



c) La CPU invia il dato alla Periferica.



d) La CPU invia il segnale Uscita Pronta alla Periferica (questo può essere un collegamento hardware).

Figura 11-6. Un Handshake di Uscita

INTERVALLI DI TEMPO (RITARDI)

USI DEGLI INTERVALLI DI TEMPO

Un problema che affronteremo completamente nelle considerazioni d'ingresso/uscita è la generazione di intervalli di tempo di lunghezze specifiche. Tali intervalli sono necessari per eliminare i rimbalzi degli interruttori meccanici (per smorzare le loro transizioni irregolari), per fornire impulsi di lunghezze e frequenze specifiche per display e per fornire temporizzazioni per dispositivi che trasferiscono dati regolarmente (per esempio, una telescrivente che trasmetta o riceva un bit ogni 9,1 ms).

METODI PER PRODURRE INTERVALLI DI TEMPO

Possiamo produrre intervalli di tempo in molti modi:

1) In hardware con monostabili o multivibratori monostabili. Questi dispositivi producono un singolo impulso di durata fissata in risposta ad un impulso di ingresso.

2) Con una combinazione di hardware e software con un temporizzatore programmabile in modo flessibile come il circuito Contatore-Temporizzatore Z80 (o CTC) per microcomputer basati sullo Z80, come descritto in An Introduction to Microcomputers: Volume 2 – Some Real Microprocessors. Il CTC può fornire intervalli di tempo di varie lunghezze con una varietà di condizioni di inizio e di fine.

3) In software con programmi di ritardo. Questi programmi impiegano il processore come un contatore. Ciò è possibile perchè il processore ha come riferimento un clock stabile, ma tale metodo chiaramente sotto-utilizza il processore. Tuttavia, i programmi di ritardo non richiedono hardware aggiuntivo e spesso impiegano tempo del processore che altrimenti andrebbe perso.

SCELTA DI UN METODO DI TEMPORIZZAZIONE

La scelta tra questi metodi dipende dalla vostra applicazione. Il metodo software non è costoso, ma può sovraccaricare il processore. I temporizzatori programmabili sono relativamente costosi, ma sono facili da interfacciare e possono essere in grado di gestire un gran numero di complessi compiti di temporizzazione.

ROUTINE DI RITARDO

RITARDO ELEMENTARE IN SOFTWARE

Un semplice programma di ritardo lavora come segue:

Passo 1) Caricare in un registro un valore specificato.
Passo 2) Decrementare il registro.
Passo 3) Se il risultato del Passo 2 non è zero, ripetere il Passo 2.

Questo programma fa solo uso del tempo. La quantità di tempo utilizzata dipende dal tempo di esecuzione delle varie istruzioni. La massima lunghezza di ritardo è limitata dalla dimensione del registro; tuttavia, l'intera routine può essere posta all'interno di una routine simile che impieghi un altro registro, e così via.

ROUTINE DI RITARDO TRASPARENTE

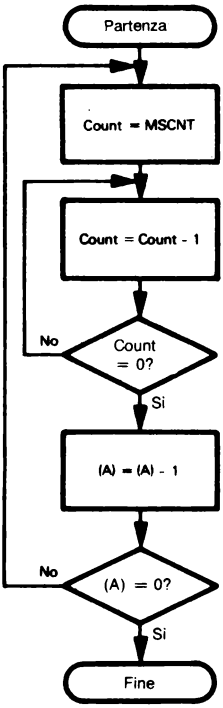
L'esempio seguente impiega il Registro C e l'Accumulatore per fornire ritardi sino a 225 ms. La scelta dei registri è arbitraria. Noi possiamo, infatti trovare l'impiego di una coppia di registri (es. BC) più conveniente. Una istruzione PUSH BC all'inizio del programma di ritardo ed un POP BC alla fine farà si che il programma non influenzerà nessun registro. Un tale programma è detto essere «trasparente» al programma chiamante. Si noti che le istruzioni PUSH E POP devono essere incluse nelle frazioni di tempo.

ESEMPIO

Programma di ritardo utilizzante gli Accumulatori

Scopo: Il programma fornisce un ritardo di 1 ms per il contenuto dell'Accumulatore B.

Diagramma di Flusso:



Il valore di MSCNT dipende dalla velocità della CPU e dal ciclo di memoria.

Programma Sorgente:

```
DELAY:  LD      C,MSCNT     ;POSIZIONA IL CONTATORE PER 1 MS DI RITARDO
DLY1:   DEC     C           ;CONTATORE = CONTATORE -1
        JR      NZ,DLY1     ;CONTINUA FINCHÈ CONTATORE = ZERO
        DEC     A           ;DECREMENTA IL NUMERO DEI MS RIMANENTI
        JR      NZ,DELAY    ;CONTINUA SINCHÈ IL NUMERO DEI MS = ZERO
        RET
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Operativo)		
0030	0E	DELAY:	LD	C,MSCNT
0031	MSCNT			
0032	0D	DLY1:	DEC	C
0033	20		JR	NZ,DLY1
0034	FD			
0035	3D		DEC	A
0036	20		JR	NZ,DELAY
0037	F8			
0038	C9		RET	

Bilancio del tempo

Istruzione		Numero di volte eseguita
LD	C,MSCNT	(A)
DEC	C	(A) x MSCNT
JR	NZ,DLY1	(A) x MSCNT
DEC	A	(A)
JR	NZ,DELAY	(A)
RET		

Il tempo totale impiegato sarebbe (A) x 1 ms. Se la memoria sta funzionando alla massima velocità, le istruzioni richiedono il seguente numero di cicli di clock.

LD	C,MSCNT	7
DEC	C o DEC A	4
JR	NZ	7 o 12
RET		10

Il numero di volte alternativo per JR vale a seconda che la condizione venga soddisfatta (12) o non venga soddisfatta (7).

Ignorando le istruzioni CALL e RET (che si presentano una sola volta), il programma richiede:

$$(A) \times (7 + 16 \times MSCNT - 5 + 16) - 5$$

cicli di clock. I «-5» derivano dal fatto che JR richiede un tempo minore durante il ciclo finale quando la condizione non è soddisfatta.

COSTANTE DEL LOOP DI RITARDO CON Z80

In tal modo, per ottenere il ritardo di 1 ms, si ha

$$13 + 16 \times MSCNT = Nc$$

dove Nc è il numero di cicli di clock per millisecondo. Alla frequenza di clock standard di 4 MHz per lo Z80, Nc = 4000, perciò:

$$16 \times MSCNT = 3987$$

MSCNT = 249 (F9 in esadecimale) ad una frequenza di clock di 4MHz per lo Z80

SEMPLICI DISPOSITIVI DI I/O

IL DISPOSITIVO PARALLELO DI INGRESSO/USCITA DELLO Z80 (PIO)

L'elemento chiave nella maggior parte delle sezioni di ingresso/uscita dello Z80 è il Circuito Parallelo Z80 d'Ingresso/Uscita o PIO. Questo dispositivo riunisce in sé latch, buffer, flip-flop, ed altri circuiti logici necessari per l'«handshake» e per altre semplici tecniche di interfacciamento. Il PIO contiene molte connessioni logiche, alcuni insiemi di queste possono essere selezionate conformemente ai contenuti di registri programmabili. Pertanto, il progetto ha sotto il suo controllo l'equivalente di un «Circuit Designer's Casebook». La fase di inizializzazione del programma pone i valori appropriati nei registri in modo da selezionare le connessioni logiche richieste. Una sezione d'ingresso/uscita basata sul PIO può gestire molte diverse applicazioni e si possono fare cambiamenti o correzioni in software piuttosto che con nuovi collegamenti.

La Figura 11-7 è lo schema a blocchi di un PIO. Il dispositivo contiene due porte ad 8 bit pressoché identiche — A, che è di solito una porta d'ingresso, e B, che è usualmente una porta d'uscita. Ciascuna porta (vedi Figura 11-B) contiene:

LINEE DI CONTROLLO E REGISTRI DEL PIO

- Un registro ad 8 bit d'Uscita dei Dati.
- Un registro ad 8 bit d'Ingresso dei Dati.
- Un registro a 2 bit di Controllo del Modo, che indica se la porta è d'uscita, d'ingresso, bidirezionale, o del modo di controllo.
- Un registro ad 8 bit di Controllo di Ingresso/Uscita, che determina se i corrispondenti pin (morsetti) dei dati sono ingressi (1) o uscite (0) nel modo di controllo.
- Due linee di controllo (STB e ROY) che sono configurate nel registro del Modo di Controllo. Queste linee possono essere impiegate per i segnali di handshake mostrati nelle Figure 11-5 e 11-6.
- Un registro a 2 bit di Controllo di Maschere (impiegato solo nel modo di controllo) che determina la polarità attiva degli ingressi e se essi sono logicamente messi in OR o in AND per formare un segnale di interrupt.
- Un registro ad 8 bit Maschera (impiegato soltanto nel modo di Controllo) che determina quali linee della porta saranno controllate per formare il segnale di interrupt.
- Un registro ad 8 bit di Indirizzo di Vettore usato con il sistema di interrupt.

Per ora, ci interesseremo soltanto dei registri di Controllo del Modo, dei registri di Controllo d'Ingresso/Uscita, e delle linee di controllo. Tratteremo le caratteristiche relative all'interrupt del PIO nel Capitolo 1-2.

I significati dei bit nei vari registri di controllo e di maschera sono collegati all'hardware relativo e sono completamente arbitrari in quanto hanno a che fare con il programmatore in linguaggio assembly. Dobbiamo o memorizzarli o cercarli in questo capitolo e nel Capitolo 1-2.

INDIRIZZI DEL PIO

Ciascun PIO occupa quattro indirizzi, per le porte d'ingresso e quattro indirizzi per le porte d'uscita. Le linee B/A SEL (scelta della Porta B o A) e C/D SEL (scelta del Controllo o del Dato) scelgono una delle quattro porte come descritto nella Tabella 11-1. Molto spesso, i progettisti collegano il bit A_0 dell'indirizzo all'ingresso B/A SEL ed il bit A_1 dell'indirizzo all'ingresso C/D SEL. Il PIO allora occupa quattro indirizzi di porte consecutivi come descritto nell'ultima colonna della Tabella 11-1.

Chiaramente ci sono più registri interni di controllo che indirizzi di porta ad essi riservati. Infatti, tutti i registri di controllo per ciascuna porta occupano un indirizzo in accordo alla connessione C/D SEL. Di conseguenza alcuni bit del dato inviato ad un registro di controllo sono realmente usati per l'indirizzamento. Notiamo le situazioni seguenti (vedere Tabella 11-2):

$D_0 = 0$ vuol dire che i rimanenti bit di dato sono caricati nel registro del Vettore di Interrupt.

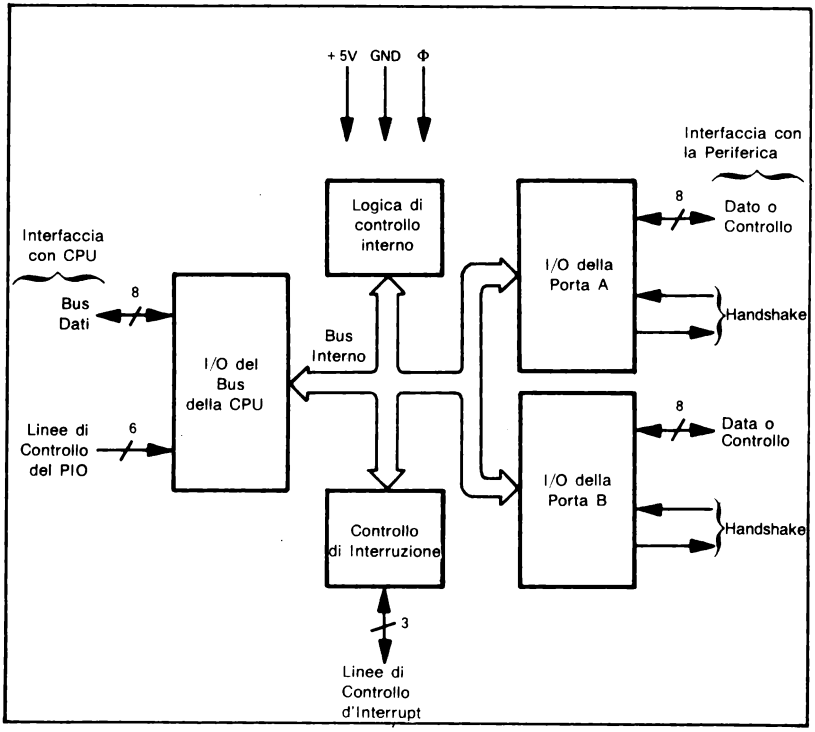


Figura 1-7. Schema a Blocchi del PIO
(Per gentile concessione della Zilog)

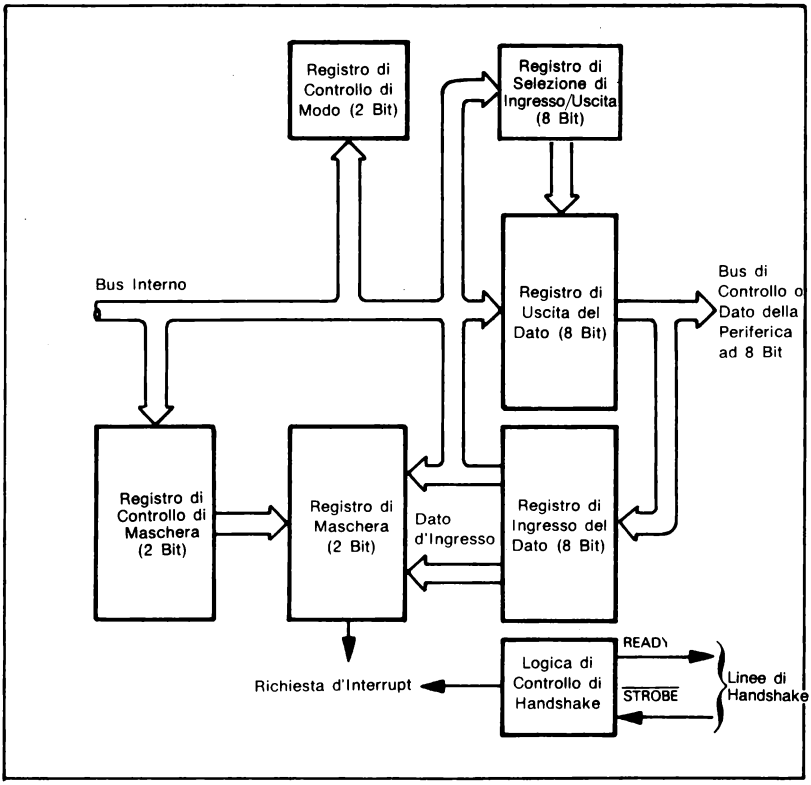


Figura 11-8. Schema a Blocchi della Porta del PIO
(Per gentile concessione della Zilog)

Tabella 11-1. Indirizzi del PIO

CONTROLLO O SELEZIONE DEL DATO	SELEZIONE DELLA PORTA A O B	REGISTRO INDIRIZZATO	INDIRIZZO DI PORTA (A PARTIRE DA PIOADD)
0	0	Registro di Dato A	PIOADD
0	1	Registro di Dato B	PIOADD+1
1	0	Controllo A	PIOADD+2
1	1	Controllo B	PIOADD+3
Gli indirizzi di porta presuppongono che C/D SEL sia collegato ad A_1 e B/A SEL ad A_0.			

Tabella 11-2. Indirizzamento dei Registri di Controllo del PIO

REGISTRO	INDIRIZZAMENTO
CONTROLLO DI MODO	$D_3 = D_2 = D_1 = D_0 = 1$
CONTROLLO DI I/O	LA SUCCESSIVA PAROLA DOPO IL CONTROLLO DEL MODO POSIZIONA IL MODO 3
REGISTRO DI CONTROLLO DI MASCHERA	$D_3 = 0$, $D_2 = D_1 = D_0 = 1$
REGISTRO DI MASCHERA D'INTERRUPT	LA PROSSIMA PAROLA DOPO IL REGISTRO DI CONTROLLO DI MASCHERA È RAGGIUNTA CON $D_4 = 1$
ABILITAZIONE DELL'INTERRUPT	$D_3 = D_2 = 0$, $D_1 = D_0 = 1$
VETTORE D'INTERRUPT	$D_0 = 1$

$D_3 = 0$, $D_2 = D_1 = D_0 = 1$ vuol dire che i rimanenti bit di dato sono caricati nel registro di Maschera di Controllo. Se $D_4 = 1$ la successiva parola di controllo è caricata nel registro di Maschera di Interrupt. Gli Interrupt possono essere abilitati o disabilitati con $D_3 = D_2 = 0$, $D_1 = D_0 = 1$.

$D_3 = D_2 = D_1 = D_0 = 1$ vuol dire che i rimanenti bit di dato sono caricati nel registro di Controllo del Modo. Se $D_7 = D_6 = 1$ (modo di controllo), la successiva parola di controllo è caricata nel registro di Controllo d'Ingresso/Uscita.

Questa suddivisione di un indirizzo esterno vuol dire che:

1) Il programmatore deve stare molto attento all'ordine delle operazioni. Il significato di una particolare istruzione di Output dipende dalla sequenza con la quale essa si presenta.
2) Il programmatore documenterà la configurazione del PIO in dettaglio. Il dispositivo è complesso ed è improbabile che un lettore sia in grado di dare molto significato al di là della sequenza di operazioni che lo configurano.

Noteremo che di solito si rappresentano i registri di controllo del PIO soltanto una sola volta in fase di inizializzazione del programma. La parte restante del programma pertanto impiega solo i registri del PIO.

CONTROLLO DEL MODO DEL PIO

MODI DEL PIO

Il modo di operare di un Pio è stabilito dalla scrittura di una parola di controllo nel PIO nella forma mostrata in Figura 11-1. La Tabella 11-3 descrive i significati dei vari modi e le parole di controllo richieste per determinarli. Si noti che i bit D_5 e D_4 non sono impiegati. Alla occasione, il PIO si posiziona nel modo 1 (ingresso).

Possiamo riassumere i modi come segue:

1) Modo 0 — USCITA

MODI DI USCITA DEL PIO

La scrittura del dato all'interno del registro della porta di Uscita vi mantiene il dato che appare sulla porta del Data Bus. La linea READY (RDY) va alta ad indicare Dato Pronto; essa rimane alta sinché la periferica non spedisce un fronte di salita sulla linea STROBE ($\overline{STB}$) per indicare Dato Accettato o Dispositivo Pronto. Il fronte di salita di $\overline{STB}$ provoca un interrupt se l'interrupt è stato abilitato.

2) Modo 1 — INGRESSO

MODO D'INGRESSO DEL PIO

La periferica memorizza il dato nel registro della porta d'Ingresso impiegando il segnale STROBE. Il fronte di salita di $\overline{STB}$ provoca un interrupt (se abilitato) e disattiva RDY. Quando la CPU legge il dato, RDY va alto per indicare Dato Accettato o Registro d'Ingresso vuoto. Si noti che la periferica può fare uno «strobe» del dato all'interno del registro senza curarsi dello stato di RDY. Il programmatore deve allora gestire il problema di overtun, cioè il posizionamento di un nuovo dato nel registro prima che sia stato letto il dato vecchio.

3) Modo 2 — BIDIREZIONALE

MODO BIDIREZIONALE DEL PIO

Questo modo impiega tutte e quattro le linee di «handshake», perciò è permesso solo sulla porta A. I segnali RDY e $\overline{STB}$ della porta A sono impiegati come controllo di uscita ed i segnali RDY e $\overline{STB}$ della porta B sono impiegati come controllo d'Ingresso. La sola differenza tra questo modo e una combinazione dei modi 0 e 1 è che il dato del Registro d'Uscita della Porta A è abilitato verso il Bus Dati della porta solo quando $\overline{ASTB}$ è attivo. Ciò permette al bus della Porta A di essere impiegato bidirezionalmente sotto il controllo di $\overline{ASTB}$ (Richiesta di Dati d'Uscita) e $\overline{BSTB}$ (Disponibilità di Dati d'Ingresso). Si noti che in questo modo i segnali di controllo del lato B sono governati dal Registro di Ingresso A.

4) Modo 3 — CONTROLLO

MODO DI CONTROLLO DEL PIO

DIREZIONI DEL PIO NEL MODO DI CONTROLLO

Questo modo non impiega i segnali RDY e $\overline{STB}$. Esso è inteso per applicazioni di stato e di controllo in cui ogni bit ha un significato individuale. Quando è selezionato il modo 3, la successiva parola di controllo inviata al PIO definisce le direzioni dei bit di dato della porta (Figura 11-9). Un «1» nella posizione di un bit rende la corrispondente linea del bus un ingresso, mentre uno «0» la rende un'uscita.

Posizionamento del Modo

M1	M0	Modo
0	0	Uscita
0	1	Ingresso
1	0	Bidirezionale
1	1	Controllo dei Bit

M1	M0	X	X	1	1	1	1

Quando si sceglie il Modo 3, la parola successiva deve posizionare il Registro di I/O.

I/O7	I/O6	I/O5	I/O4	I/O3	I/O2	I/O1	I/O0

I/O = 1 Posiziona i bit in Ingresso
I/O = 0 Posiziona i bit in Uscita

Modo del PIO	Significato	Parola di Controllo	
		(Binario)	(Esadec.)
0	Uscita	00001111	0F
1	Ingresso	01001111	4F
2	Bidirezionale	10001111	8F
3	Controllo	11001111	CF

Si noti che i bit 4 e 5 non sono usati e potrebbero avere un valore qualsiasi.

Figura 11-9. Controllo del Modo per lo Z80 PIO

Si notino le seguenti caratteristiche dei modi del PIO:

CARATTERISTICHE DEI MODI DEL PIO

1) Nei modi 0, 1 e 2 la periferica indica Dato Pronto, Dispositivo Pronto o Dato Accettato, con un fronte di salita sulla linea $\overline{STB}$. Tale fronte provoca anche un interrupt se l'interrupt è abilitato.

2) Nei modi 0, 1 e 2 il PIO indica Dato Pronto, Buffer d'Ingresso Vuoto, o Dato Accettato trasmettendo RDY alto. Questo segnale resta alto sino al successivo fronte di salita su $\overline{STB}$.

3) Soltanto la Porta A può essere impiegata bidirezionalmente. Se la porta A è nel modo 2 (bidirezionale), la porta B può soltanto essere nel modo 3 (controllo), giacché non sono disponibili linee di handshake.

4) Il modo di controllo (3) è il solo modo in cui è impiegato il registro di Controllo d'Ingresso/Uscita. Altrimenti, l'intera porta è impiegata o come ingresso o come uscita.

5) Se non si impiegano gli interrupt, non c'è modo per il processore di accorgersi se si è presentato un impulso su $\overline{STB}$. Il PIO è stato progettato per essere gestito ad interrupt piuttosto che con sistemi a «polling» (si veda il Capitolo 12). $\overline{STB}$ dovrà essere mantenuto basso se o non è impiegato.

6) Il processore non può controllare direttamente le linee RDY. La linea RDY su una porta va alta quando il dato è trasferito da o verso la porta e va basso sul fronte di salita di $\overline{STB}$.

7) Il contenuto del Registro di Uscita del dato può essere letto se la porta è nel modo di uscita o nel modo bidirezionale. Se la porta è nel modo di controllo, il dato del registro di uscita può essere letto dalle linee assegnate come uscite. Il contenuto dei registri di controllo non può essere letto.

8) Se l'uscita RDY è legata all'ingresso $\overline{STB}$ su una porta nel modo di uscita, RDY andrà alta per un periodo di clock dopo ogni operazione di uscita. Questo breve impulso può essere impiegato per gestire un multiplex display come mostrato in Figura 11-1.

CONFIGURAZIONE DEL PIO

Il programma deve selezionare le connessioni logiche nel PIO prima del trasferimento di dati da o verso esso. Questa selezione (configurazione) fa di solito parte del programma di inizializzazione. Si noti che il PIO va nel modo d'ingresso all'accensione con tutti gli interrupt disabilitati ed inibiti ed i segnali di controllo disattivati (bassi). Quindi, il PIO non deve essere un ingresso di RESET e non ritorna necessariamente allo stato di reset quando la CPU subisce un reset. I passi per configurare un PIO sono:

PASSI PER CONFIGURARE UN PIO

1) Stabilire il modo di funzionamento scrivendo le appropriate parole di controllo nel registro di controllo del Modo. Si potrebbero pure inviare le informazioni relative al controllo dell'Interruzione e al modo di I/O.
2) Se si è nel modo 3, stabilire le direzioni dei pin di I/O scrivendo una parola di controllo nel registro di Controllo Ingresso/Uscita. Questa parola deve seguire la parola di controllo che ha selezionato il modo 3.

Vediamo ora alcuni esempi di configurazione di un PIO senzà interrupt:

1) PORTA DI USCITA

```
        LD      A,00001111B   ;RENDE LA PORTA B DI USCITA
        OUT     (PIOCRB),A
```

2) PORTA DI INGRESSO

```
        LD      A,01001111B   ;RENDE LA PORTA A DI INGRESSO
        OUT     (PIOCRA),A
```

3) PORTA BIDIREZIONALE

```
        LD      A,10001111B   ;RENDE LA PORTA A BIDIREZIONALE
        OUT     (PIOCRA),A
```

Si ricordi che solo la Porta A può essere bidirezionale e che allora la Porta B deve essere una porta di controllo.

4) PORTA DI CONTROLLO, TUTTI INGRESSI

```
        LD      A,11001111B   ;RENDE LA PORTA A DI CONTROLLO
        OUT     (PIOCRA),A
        LD      A,0FFH        ;TUTTI I BIT SONO DI INGRESSO
        OUT     (PIOCRA),A
```

5) PORTA DI CONTROLLO, TUTTE USCITE

```
        LD      A,11001111B   ;RENDE LA PORTA B DI CONTROLLO
        OUT     (PIOCRB),A
        SUB     A             ;TUTTI I BIT SONO DI USCITA
        OUT     (PIOCRB),A
```

6) PORTA DI CONTROLLO, LINEE 1,5,6 INGRESSI; LINEE 0,2,3,4,7 USCITE

```
        LD      A,11001111B   ;RENDE LA PORTA A DI CONTROLLO
        OUT     (PIOCRA),A
        LD      A,01100010B   ;LINEE 1,5,6, INGRESSI – 0,2,3,4,7 USCITE
        OUT     (PIOCRA),A
```

ISTRUZIONI D'INGRESSO/USCITA DELLO Z80

ISTRUZIONI DI I/O DELLO Z80

Il microprocessore Z80 ha un esteso insieme di istruzioni d'Ingresso/Uscita. Tutte le istruzioni di I/O impiegano indirizzi ad 8 bit per i dispositivi permettendo così 256 porte di ingresso e 256 porte di uscita. Ma si ricordi che ogni PIO occupa quattro indirizzi di porte di uscita e quattro indirizzi di porte di ingresso.

Le istruzioni di I/O possono essere raggruppate come segue:

1) Istruzioni che impiegano un indirizzamento assoluto. IN A, (porta) e OUT (porta), A trasferiscono un dato ad 8 bit tra l'Accumulatore e la porta indirizzata dal secondo byte dell'Istruzione.
2) Istruzioni a singolo byte che impiegano un indirizzamento indiretto mediante registro. IN reg, (C) e OUT (C), reg trasferiscono un dato ad 8 bit tra il registro specificato e la porta indirizzata dal Registro C.
3) Istruzioni di I/O di blocco. INI e OUTI trasferiscono dati ad 8 bit tra la locazione di memoria indirizzata dalla coppia di registri HL e la porta indirizzata dal Registro C. Entrambe le istruzioni poi incrementano la coppia di registri HL e decrementano il contatore di byte nel registro B. Il flag Z è posto ad 1 se B è decrementato a zero, altrimenti è posto a 0. IND ed OUTD sono istruzioni identiche se si eccettua il fatto che decrementano la coppia di Registri HL; invece di incrementarla.
4) Istruzioni di I/O di blocco ripetuto. INIR e OTIR ripetono gli effetti di INI e OUTI, rispettivamente, sinché B non è decrementato a zero. INDR e OTDR sono nella medesima relazione con IND e OUTD.

Potete notare le seguenti caratteristiche per ogni gruppo di istruzioni:

ISTRUZIONI DI I/O CON INDIRIZZAMENTO ASSOLUTO

1) Istruzioni con indirizzamento assoluto

- Il Dato è sempre trasferito da o verso l'Accumulatore.
- Non è toccato alcun flag.
- L'indirizzo delle porte è parte della memoria di programma e non può essere cambiato se la memoria è a sola lettura.

ISTRUZIONI DI I/O CON INDIRIZZAMENTO INDIRETTO

2) Istruzioni a singolo byte con indirizzamento indiretto con registro.

- Il dato può essere trasferito a o da ciascuno de registri primari ad 8 bit (A,B,C,D,E,H,L). Comunque, si ricordi che il Registro C contiene l'indirizzo della porta.
- IN reg, (C) posiziona ad 1 i flag di Segno (S), di zero (Z) e di Parità (PIO) a secondo del valore del dato di ingresso. Il flag di Riporto (C) non è modificato, ma i flag di Half Carry (H) e di Negativo (N) sono posti a 0. OUT (C), reg non influenza alcun flag.
- L'indirizzo della porta è sempre nel Registro C. Questo indirizzo non è parte della memoria di programma e potrebbe essere un parametro per una subroutine di I/O (o driver di I/O). In tal modo un driver di I/O potrebbe essere impiegato in un gran numero di differenti applicazioni o con parecchi dispositivi di I/O simili nella stessa applicazione.

DRIVER DI I/O

3) Istruzioni di I/O a blocchi

ISTRUZIONI DI I/O A BLOCCHI

- Il dato è sempre trasferito alla o dalla locazione di memoria indirizzata dalla coppia di Registri HL.
- Il flag Z (Zero) è posto ad 1 se il registro B è decrementato a zero, altrimenti è azzerato. I flag S (Segno), P/O (Parità), e H (Half Carry) sono influenzati, ma i loro valori finali sono indefiniti.
- L'indirizzo della porta è sempre nel Registro C. Di nuovo questo indirizzo potrebbe essere un parametro per un driver di I/O.
- Il registro B è un contatore ad 8 bit. Pertanto, le istruzioni di I/O a blocchi ripetute possono trasferire al massimo 256 byte. Ciò si differenzia dalla istruzione di Spostamento di un Blocco e di Confronto di un Blocco, che impiegano la coppia di registri BC come un contatore a 16 bit e che possono gestire 65K byte.

ESEMPI DI ISTRUZIONI DI I/O

Alcuni esempi delle varie istruzioni di I/O (senza alcuna considerazione sulla temporizzazione) sono:

1) Carica l'Accumulatore della Porta di Ingresso 2

a. Impiegando l'indirizzamento assoluto

```
IN      A,(2)
```

b. Impiegando un indirizzamento indiretto con registro

```
LD      C,2
IN      A,(C)
```

2) Immagazzina il contenuto dell'Accumulatore nella Porta di Uscita 5

a. Impiegando un indirizzamento assoluto

```
OUT     (5),A
```

b. Impiegando un indirizzamento indiretto con registro

```
LD      C,5
OUT     (C),A
```

3) Carica la locazione di memoria 0040 dalla Porta di Ingresso 2

a. Impiegando un indirizzamento assoluto

```
IN      A,(2)      ;PRENDI IL DATO
LD      (40H),A    ;IMMAGAZZINA IL DATO
```

b. Impiegando un indirizzamento indiretto con registro

```
LD      C,2        ;PRENDI IL NUMERO DELLA PORTA
IN      A,(C)      ;PRENDI IL DATO
LD      (40H),A    ;IMMAGAZZINA IL DATO
```

c. Impiegando istruzioni di I/O a blocchi

```
LD      C,2        ;PRENDI IL NUMERO DELLA PORTA
LD      HL,40H     ;PRENDI LA DESTINAZIONE DI MEMORIA
INI                ;PRENDI IL DATO
```

4) Immagazzina il contenuto della locazione di memoria 0040 nella Porta di Uscita 5

a. Impiegando l'indirizzamento assoluto

```
        LD      A,(40H)   ;PRENDI IL DATO
        OUT     (5),A     ;TRASMETTI IL DATO
```

b. Impiegando un indirizzamento indiretto con registro

```
        LD      C,5       ;PRENDI IL NUMERO DELLA PORTA
        LD      A,(40H)   ;PRENDI IL DATO
        OUT     (C),A
```

c. Impiegando istruzioni di I/O a blocchi

```
        LD      C,5       ;PRENDI IL NUMERO DELLA PORTA
        LD      HL,40H    ;PRENDI LA SORGENTE DI MEMORIA
        OUTI              ;TRASMETTI IL DATO
```

5) Carica le locazioni di memoria da 0040 a 0047 dalla porta di Ingresso 2

a. Impiegando l'indirizzamento assoluto

```
        LD      HL,40H    ;PRENDI L'INDIRIZZO DI PARTENZA DEL DATO
        LD      B,8       ;CONTATORE DI BYTE = 8
INBYTE: IN      A,(2)     ;ATTINGI IL BYTE DI DATO
        LD      (HL),A    ;MEMORIZZA IL BYTE IN MEMORIA
        INC     HL
        DJNZ    INBYTE
```

b. Impiegando istruzioni di I/O a blocchi

```
        LD      HL,40H    ;PRENDI L'INDIRIZZO DI PARTENZA DEL DATO
        LD      B,8       ;CONTATORE DI BYTE = 8
        LD      C,2       ;PRENDI IL NUMERO DELLA PORTA
INBYTE: INI
        JR      NZ,INBYTE
```

c. Impiegando istruzioni di I/O a blocchi ripetute

```
        LD      HL,40H    ;PRENDI L'INDIRIZZO DI PARTENZA DEL DATO
        LD      B,8       ;CONTATORE DI BYTE = 8
        LD      C,2       ;PRENDI IL NUMERO DELLA PORTA
        INIR              ;SPOSTA I BYTE DI !NGRESSO IN MEMORIA
```

6) Trasmette i contenuti delle locazioni di memoria da 0040 a 0047 alla Porta di Uscita 5.

a. Impiegando l'indirizzamento assoluto

```
        LD      HL,40H   ;PRENDI L'INDIRIZZO DI PARTENZA DEL DATO
        LD      B,8      ;PRENDI IL CONTATORE DI BYTE
OTBYTE: LD      A,(HL)   ;ATTINGI IL BYTE DALLA MEMORIA
        OUT     (5),A    ;BYTE IN USCITA
        INC     HL
        DJNZ    OTBYTE
```

b. Impiegando istruzioni di I/O a blocchi

```
        LD      HL,40H   ;PRENDI L'INDIRIZZO DI PARTENZA DEL DATO
        LD      B,8      ;PRENDI IL CONTATORE DI BYTE
        LD      C,5      ;PRENDI IL NUMERO DELLA PORTA
OTBYTE: OUTI             ;BYTE IN USCITA DALLA MEMORIA
        JR      NZ,OTBYTE
```

c. Impiegando istruzioni di I/O a blocchi ripetute

```
        LD      HL,40H   ;PRENDI L'INDIRIZZO DI PARTENZA DEL DATO
        LD      B,8      ;PRENDI IL CONTATORE DI BYTE
        LD      C,5      ;PRENDI IL NUMERO DELLA PORTA
        OTIR             ;BYTE IN USCITA DALLA MEMORIA
```

USO DELLE ISTRUZIONI DI I/O A BLOCCHI

Si noti che le istruzioni di I/O a blocchi ripetute funzionano continuamente. Non si può stabilire una qualsiasi temporizzazione tra due trasferimenti. Pertanto, queste istruzioni non possono essere impiegate a meno che la periferica funzioni alla stessa velocità del processore o la temporizzazione sia gestita separatamente in hardware. Modi per gestire la temporizzazione in hardware comprendono il forzamento del processore in stati di Wait o il buffering del dato. Si noti che tutte le istruzioni di I/O a Blocchi collocano il contenuto del contatore di byte (Registro B) nella metà più significativa del BUS INDIRIZZI durante il trasferimento di I/O in corso. Nelle operazioni di Uscita prima è decrementato il Registro B. Il valore del contatore di byte è allora disponibile per i circuiti esterni.

Un'applicazione ovvia per le istruzioni di I/O a blocchi è la configurazione dei PIO. Un gran numero di parole devono spesso essere inviate ad un registro di controllo per determinare il modo di funzionamento per scegliere le direzioni dei pin, e per stabilire il sistema d'interrupt. Non si pongono problemi di temporizzazione, poiché i PIO funzionano alla stessa velocità della CPU. Tratteremo in seguito in questo capitolo e nel Capitolo 12 le configurazioni dei PIO Z80 e delle interfacce seriali (SIO) con istruzioni di I/O a Blocchi.

Negli esempi di I/O che seguono, impiegheremo principalmente le istruzioni con indirizzamento assoluto. Potete facilmente sostituire le istruzioni con indirizzamento indiretto da registro purché vi ricordiate di inizializzare il Registro C. Occasionalmente indicheremo delle applicazioni per istruzioni di I/O a blocchi.

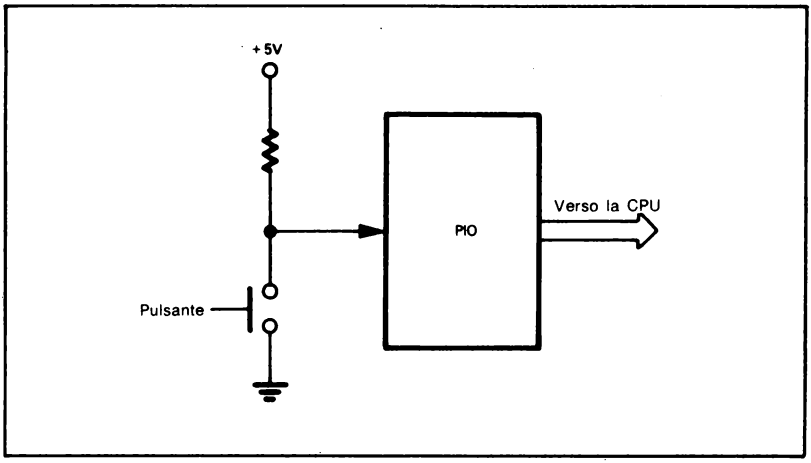


Figura 11-10. Un Circuito a Pulsante

ESEMPI

Un Interruttore a Pulsante

Scopo: Interfacciare un singolo interruttore a pulsante (o un single-pole, single-throw (SPST) switch: interruttore unipolare ad una via) con un microprocessore Z80. Il pulsante è un interruttore meccanico che fornisce una chiusura a singolo contatto (cioè uno zero logico quando è premuto).

Schema del Circuito:

La Figura 11-10 mostra il circuito richiesto per interfacciare il pulsante. Esso utilizza un solo bit di un PIO Z80 che agisce come un buffer; non è necessaria alcuna memorizzazione, poiché il pulsante resta chiuso per molti cicli di clock della CPU. Premendo il pulsante, il bit di ingresso del PIO va basso. La resistenza di pullup assicura che il bit di ingresso è alto se il pulsante non viene premuto.

Esempi di programmazione:

Eseguiremo due compiti con questo circuito, che sono:

a) Posizionare ad 1 una locazione di memoria in funzione dello stato del pulsante.
b) Contare quante volte il pulsante viene premuto.

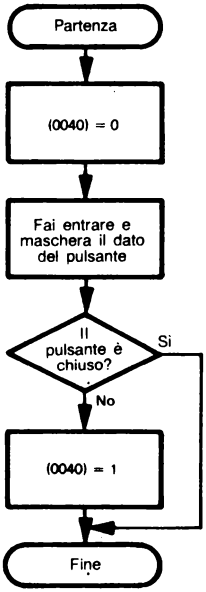
Compito 1: Determinare la chiusura dell'interruttore.

Scopo: Posizionare la locazione di memoria 0040 ad uno se il pulsante non viene premuto e a zero se viene premuto.

Casi campione:

1) Pulsante aperto (cioè non premuto)
   Risultato = (0040) = 01
2) Pulsante chiuso (cioè premuto)
   Risultato = (0040) = 00

Diagramma di Flusso:



Programma Sorgente:

```
        LD      A,01001111B   ;RENDI LA PORTA A UN INGRESSO
        OUT     (PIOCRA),A
        LD      HL,40H        ;INDICATORE = 0
        LD      (HL),0
        IN      A,(PIODRA)    ;LEGGI LA POSIZIONE DEL PULSANTE
        AND     MASK          ;IL PULSANTE È CHIUSO (0)?
        JR      Z,DONE        ;SÌ, VAI A DONE
        INC     (HL)          ;NO, INDICATORE = 1
DONE:   HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Operativo)	
0000	3E	LD	A,01001111B
0001	4F		
0002	D3	OUT	(PIOCRA),A
0003	PIOCRA		
0004	21	LD	HL,40H
0005	40		
0006	00		
0007	36	LD	(HL),0
0008	00		
0009	DB	IN	A,(PIODRA)
000A	PIODRA		
000B	E6	AND	MASK
000C	MASK		
000D	28	JR	Z,DONE
000E	01		
000F	34	INC	(HL)
0010	76	HALT	

Gli indirizzi della porta PIOCRA e PIODRA dipendono da come il PIO è connesso nel vostro microcomputer. Le linee di controllo del PIO non sono impiegate in questo esempio. Infatti, potremmo posizionare le porte A del PIO nel modo di controllo con la sequenza di partenza:

```
LD      A,11001111B   ;RENDI LA PORTA A UN CONTROLLO
OUT     (PIOCRA),A
LD      A,0FFH        ;TUTTI I BIT DI INGRESSO
OUT     (PIOCRA),A
```

MASK dipende dal bit a cui il pulsante è collegato; essa ha un uso in corrispondenza della posizione del pulsante e degli zeri delle altre posizioni.

Posizione del Tasto (Numero del Bit)	Maschera	
	Binario	Esadecimale
0	00000001	01
1	00000010	02
2	00000100	04
3	00001000	08
4	00010000	10
5	00100000	20
6	01000000	40
7	10000000	80

Se il pulsante è collegato al bit zero o al bit 7 della porta d'ingresso, il programma può impiegare una istruzione di shift, per porre ad 1 il carry e di conseguenza determinare lo stato del pulsante. Per esempio:

```
Bit 7
IN      A,(PIODRA)    ;LEGGI LA POSIZIONE DEL TASTO
RLA                   ;IL TASTO È CHIUSO (0)?
JR      NC,DONE       ;SÌ, VAI A DONE
Bit 0
IN      A,(PIODRA)    ;LEGGI LA POSIZIONE DEL TASTO
RRA                   ;IL TASTO È CHIUSO (0)?
JR      NC,DONE       ;SÌ, VAI A DONE
```

La procedura per il bit 7 è ancora più semplice se abbiamo l'indirizzo del registro dati del PIO nel Registro C. Ciò, perchè le istruzioni di ingresso utilizzanti un indirizzamento indiretto da registro (esempio IN A,(C)), influenzano il flag di Segno. La sequenza richiesta è:

```
Bit 7 (PIODRA nel Registro C)
IN      A,(C)       ;LEGGI LA POSIZIONE DEL TASTO
JP      P,DONE      ;VAI A DONE SE IL TASTO È CHIUSO (ZERO)
```

Se il pulsante è collegato ai bit 6 o 7 della porta di ingresso, il programma può impiegare il bit di Segno per determinare lo stato del pulsante. Per esempio:

```
Bit 7
IN      A,(PIODRA)  ;LEGGI LA POSIZIONE DEL TASTO
AND     A           ;IL TASTO È CHIUSO (ZERO)?
JP      P,DONE      ;SÌ, VAI A DONE
```

IN A, (porta) non influenza i flag; pertanto, dobbiamo utilizzare la istruzione AND A per posizionare ad 1 i flag senza modificare l'Accumulatore.

```
Bit 6
IN      A,(PIODRA)  ;LEGGI LA POSIZIONE DEL TASTO
ADD     A,A         ;IL TASTO È CHIUSO (ZERO)?
JP      P,DONE      ;SÌ, VAI A DONE
```

RLA non può essere impiegato, poiché non influenza il bit di Segno.

Compito 2: Contare le chiusure dell'interruttore.

Scopo: Contare il numero delle chiusure del pulsante incrementando la locazione di memoria 0040 dopo ogni chiusura.

Caso Campione:

Premendo il pulsante dieci volte dopo la partenza del programma dovremmo trovare

(0040) = 0A

RIMBALZO DI UN INTERRUTTORE

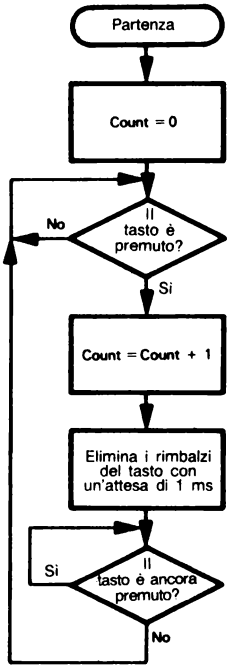
Nota: Per contare quante volte il pulsante è stato premuto, dobbiamo essere sicuri che ogni chiusura provochi una sola transizione. Tuttavia, un pulsante meccanico non produce una sola transizione per ogni chiusura, poiché i contatti meccanici rimbalzano avanti e indietro prima di stabilizzarsi nella loro posizione finale. Si può impiegare un monostabile per eliminare il rimbalzo o realizzarlo in software.

ELIMINAZIONE DEI RIMBALZI IN SOFTWARE

Il programma può eliminare il rimbalzo del pulsante attendendo dopo che ha trovato una chiusura. Il ritardo richiesto è chiamato tempo di eliminazione del rimbalzo e fa parte delle specifiche del pulsante. Tipicamente tale tempo è dell'ordine di alcuni millisecondi. Il programma non dovrebbe esaminare il pulsante durante questo periodo, poiché potrebbe confondere i rimbalzi con nuove chiusure. Il programma può sia entrare in una routine di ritardo come quella descritta in precedenza sia realizzare semplicemente altri compiti durante la quantità di tempo specificate.

Anche dopo l'eliminazione dei rimbalzi, il programma deve mettersi ancora in attesa che la chiusura presente finisca prima di andare ad esaminare una nuova chiusura. Questa procedura evita un doppio conteggio. Il programma seguente impiega un ritardo di 1ms realizzato in software per annullare il rimbalzo del pulsante. Potete provare a variare il ritardo o ad eliminarlo completamente e vedere cosa succede. Per eseguire questo programma dovete pure mettere la subroutine del ritardo nella memoria e partire dalla locazione 0030.

Diagramma di Flusso:



Programma Sorgente:

```
        LD      A,01001111B  ;RENDI LA PORTA A UN INGRESSO
        OUT     (PIOCRA),A
        LD      HL,40H
        LD      (HL),0       ;CONTEGGIO DELLE CHIUSURE = ZERO
CHKCL:  IN      A,(PIODRA)   ;LEGGI LA POSIZIONE DEL TASTO
        AND     MASK         ;IL TASTO È PREMUTO (0)?
        JR      NZ,CHKCL     ;NO, ASPETTA FINCHÈ NON LO SIA
        INC     (HL)         ;SÌ, INCREMENTA IL CONTEGGIO DELLE CHIUSURE
        CALL    DELAY        ;ASPETTA 1 MS PER ELIMINARE I RIMBALZI
CHKOP:  IN      A,(PIODRA)   ;LEGGI LA POSIZIONE DEL TASTO
        AND     MASK         ;IL TASTO È ANCORA PREMUTO (0)?
        JR      Z,CHKOP      ;SÌ, ASPETTA IL RILASCIO
        JR      CHKCL        ;NO, CERCA LA PROSSIMA CHIUSURA
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Operativo)		
0000	3E		LD	A,01001111B
0001	4F			
0002	D3		OUT	(PIOCRA),A
0003	PIOCRA			
0004	21		LD	HL,40H
0005	40			
0006	00			
0007	36		LD	(HL),0
0008	00			
0009	DB	CHKCL:	IN	A,(PIODRA)
000A	PIODRA			
000B	E6		AND	MASK
000C	MASK			
000D	20		JR	NZ,CHKCL
000E	FA			
000F	34		INC	(HL)
0010	CD		CALL	DELAY
0011	30			
0012	00			
0013	DB	CHKOP:	IN	A,(PIODRA)
0014	PIODRA			
0015	E6		AND	MASK
0016	MASK			
0017	28		JR	Z,CHKOP
0018	FA			
0019	18		JR	CHKCL
001A	EE			

Le tre istruzioni che cominciano con la label CHKOP sono usate per determinare quando l'interruttore si riapre.

Chiaramente non abbiamo veramente bisogno di un PIO per questa semplice interfaccia. Un buffer tri-state indirizzabile svolgerebbe il compito con un costo di gran lunga più basso.

UN INTERRUTTORE DI TIPO «TOGGLE»

Scopo: Interfacciare un interruttore di tipo toggle, (a ginocchiera) single-pole, double-throw (SPDT) (unipolare, a due vie) ad un microprocessore Z80. Il «toggle» è un dispositivo meccanico che è o in posizione normalmente chiusa (NC) o in posizione normalmente aperta (NO).

Schema del circuito:

ELIMINAZIONE DEI RIMBALZI CON PORTE NAND MUTUAMENTE ACCOPPIATE

La Figura 11-1 mostra la circuiteria richiesta per interfacciare l'interruttore. Come il pulsante, l'interruttore usa un solo bit di un PIO Z80 che serve come buffer indirizzabile. Diversamente dal pulsante, l'interruttore deve essere lasciato nell'una o nell'altra posizione. I tipici compiti del programma sono determinare la posizione dell'interruttore e vedere se la posizione è cambiata. O un monostabile con un impulso di lunghezza pari a pochi millisecondi o una coppia di porte NAND accoppiate l'una con l'altra (vedi Figura 11-2) possono eliminare il rimbalzo dell'interruttore meccanico.

I circuiti produrranno un solo gradino o un impulso in risposta a un cambiamento della posizione dell'interruttore anche se l'interruttore rimbalza prima di assestarsi nella sua nuova posizione.

Esempi di Programmazione:

Eseguiremo due compiti possibili con questo circuito. Essi sono:

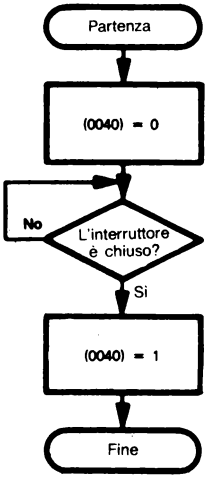
1) Posizionare ad uno una locazione di memoria quando l'interruttore è chiuso.
2) Posizionare a uno una locazione di memoria quando lo stato dell'interruttore cambia.

Compito 1: Attesa per la chiusura dell'interruttore.

Scopo: La locazione di memoria 0040 è zero sino a quando l'interruttore non si chiude, e allora è posizionata ad uno; cioè il processore azzera la locazione di memoria 0040, attende che l'interruttore sia chiuso e quindi posiziona ad uno la locazione di memoria 0040.

L'interruttore potrebbe essere denominato Run/Halt, poiché il processore non procederà sino a che l'interruttore non sia chiuso.

Diagramma di Flusso:



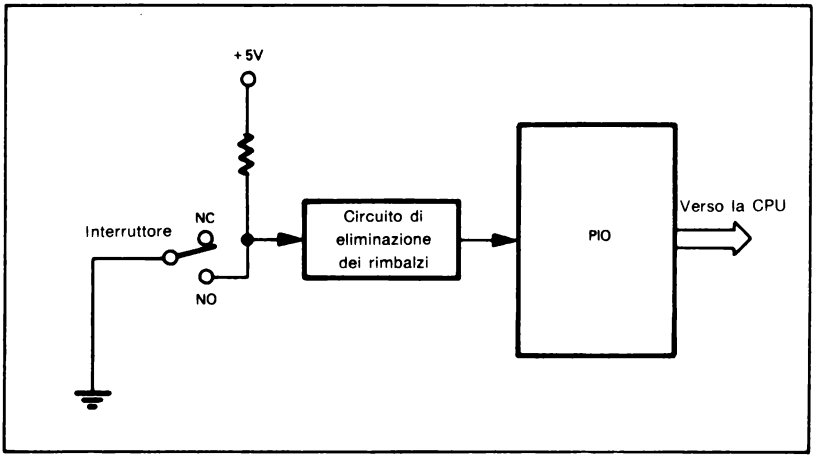


Figura 11-11. Un Circuito Switch

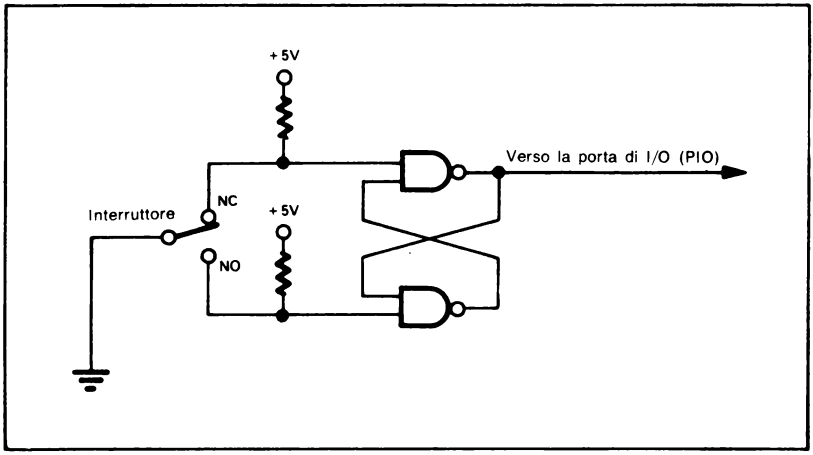


Figura 11-12. Un Circuito di Eliminazione Rimbalzo basato su Porte NAND Ad Accoppiamento-Incrociato

Programma Sorgente:

```
        LD      A,01001111B   ;RENDI LA PORTA AD UN INGRESSO
        OUT     (PIOCRA),A
        LD      HL,40H
        LD      (HL),0        ;INDICATORE = ZERO
WAITC:  IN      A,(PIODRA)    ;LEGGI LA POSIZIONE DELL'INTERRUTTORE
        AND     MASK          ;L'INTERRUTTORE È CHIUSO (0)?
        JR      NZ,WAITC      ;NO, ASPETTA CHE L'INTERRUTTORE SIA CHIUSO
        INC     (HL)          ;SÌ, INDICATORE = 1
        HALT
```

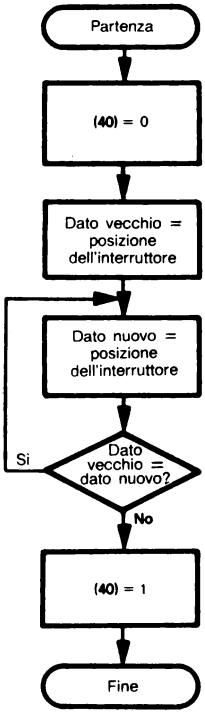
Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Operativo)		
0000	3E		LD	A,01001111B
0001	4F			
0002	D3		OUT	(PIOCRA),A
0003	PIOCRA			
0004	21		LD	HL,40H
0005	40			
0006	00			
0007	36		LD	(HL),0
0008	00			
0009	DB	WAITC:	IN	A,(PIODRA)
000A	PIODRA			
000B	E6		AND	MASK
000C	MASK			
000D	20		JR	NZ,WAITC
000E	FA			
000F	34		INC	(HL)
0010	76		HALT	

Compito 2: Attesa per il cambiamento della posizione dell'interruttore.

Scopo: La locazione di memoria 0040 resta a zero fino a che la posizione dell'interruttore non cambia; cioè, il processore attende finché l'interruttore non cambi posizione, quindi posiziona ad 1 la locazione di memoria 0040.

Diagramma di Flusso:



Programma Sorgente:

```
        LD      A,01001111B   ;RENDI LA PORTA A UN INGRESSO
        OUT     (PIOCRA),A
        LD      HL,40H
        LD      (HL),0        ;INDICATORE = ZERO
        IN      A,(PIODRA)    ;PRENDI LA POSIZIONE DEL VECCHIO INTERRUTTORE
        AND     MASK
        LD      B,A
SRCH:   IN      A,(PIODRA)    ;PRENDI LA POSIZIONE DEL NUOVO INTERRUTTORE
        AND     MASK
        CP      B             ;LE POSIZIONI NUOVA E VECCHIA SONO UGUALI?
        JR      Z,SRCH        ;SÌ, ASPETTA
        INC     (HL)          ;NO, INDICATORE = UNO
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)		Istruzione (Codice Operativo)	
0000	3E		LD	A,01001111B
0001	4F			
0002	D3		OUT	(PIOCRA),A
0003	PIOCRA			
0004	21		LD	HL,40H
0005	40			
0006	00			
0007	DB		IN	A,(PIODRA)
0008	PIODRA			
0009	E6		AND	MASK
000A	MASK			
000B	47		LD	B,A
000C	DB	SRCH:	IN	A,(PIODRA)
000D	PIODRA			
000E	E6		AND	MASK
000F	MASK			
0010	B8		CP	B
0011	28		JR	Z,SRCH
0012	F9			
0013	34		INC	(HL)
0014	76		HALT	

Un Subtract o un Exclusive-OR potrebbero sostituire Compare nel programma. Entrambe queste istruzioni, comunque, modificherebbero il contenuto dell'Accumulatore. L'OR-Esclusivo sarebbe utile se parecchi interruttori fossero collegati allo stesso PIO, in quanto esso produrrebbe un bit ad 1 per ogni interruttore che cambi stato. Come riscrivereste il programma in modo tale da annullare il rimbalzo dell'interruttore in software?

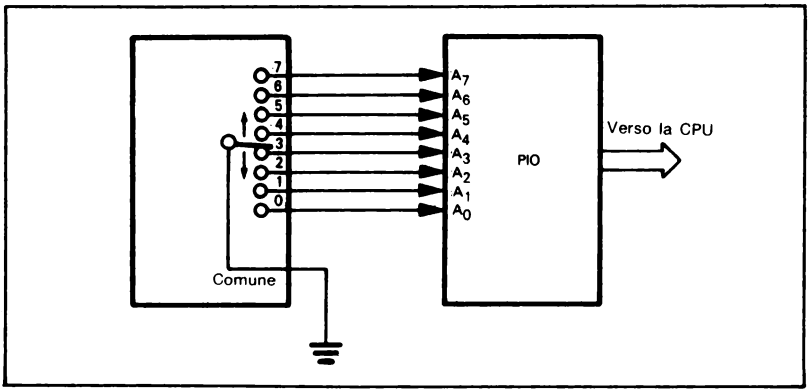


Figura 11-13. Uno Switch a Posizioni Multiple

Interruttore a posizione multipla (a rotazione, a Selettore o a Thumbwheel)

Scopo: Interfacciare un interruttore a posizione multipla con un microprocessore. Il terminale corrispondente alla posizione dell'interruttore è a terra, mentre gli altri terminali sono alti (uno logico).

Schema del circuito:

La Figura 11-13 mostra la circuiteria richiesta per interfacciare un interruttore ad 8 posizioni. L'interruttore impiega tutti e otto i bit di dato di una parte del PIO. Compiti tipici sono determinare la posizione dell'interruttore e controllare se la posizione è cambiata o meno. Si devono gestire due situazioni speciali:

1) L'interruttore è temporaneamente tra le due posizioni in modo tale che nessuno dei terminali è a massa.
2) L'interruttore non ha ancora raggiunto la sua posizione finale.

La prima di queste situazioni può essere trattata aspettando sino a che l'ingresso presenta tutti uno, cioè fino a che un terminale dell'interruttore è a massa. Possiamo trattare la seconda situazione esaminando nuovamente l'interruttore dopo un ritardo (quale 1 o 2 secondi) ed accettando l'ingresso solo quando esso rimane lo stesso. Questo ritardo non influenzerà la sensibilità del sistema all'interruttore. Possiamo anche impiegare un altro interruttore (cioè un interruttore «Load») per informare il processore di quando l'interruttore a selettore dovrebbe essere letto.

Esempi di Programmazione:

Compiremo due compiti relativi al circuito di Figura 11-13. Questi sono:

a) Controllare l'interruttore sino a che non sia in una posizione definita, quindi determinare la posizione e memorizzare il suo valore binario in una locazione di memoria.
b) Attendere che la posizione dell'interruttore cambi, quindi memorizzare la nuova posizione in una locazione di memoria.

Se l'interruttore è in una posizione, il terminale relativo a quella posizione è a massa attraverso la linea comune. Le resistenze di «pullup» sulle linee di ingresso evitano problemi provocati da rumore.

Tabella 11-3. Ingresso dei Dati rispetto alla posizione dell'interruttore

Posizione dell'Interruttore	Ingresso del Dato	
	Binario	Esadecimale
0	11111110	FE
1	11111101	FD
2	11111011	FB
3	11110111	F7
4	11101111	EF
5	11011111	DF
6	10111111	BF
7	01111111	7F

Compito 1: Determinare la posizione dell'interruttore.

Scopo: Il programma attende che l'interruttore sia in una posizione specifica ed allora sistema il numero corrispondente a quella posizione nella locazione di memoria 0040.

La Tabella 11-3 contiene gli ingressi del dato corrispondenti alle varie posizioni dell'interruttore.

USO DI UN CODIFICATORE TTL

Questa schematizzazione è inefficiente, poiché richiede otto bit per distinguere otto differenti posizioni. Un codificatore TTL o MOS potrebbe ridurre il numero di bit necessari. La Figura 11-4 mostra un circuito che impiega il codificatore TTL da 8 a 3 74LS148[4]. Colleghiamo le uscite dell'interruttore in ordine inverso, poiché il dispositivo 74LS148 ha ingressi e uscite attivi bassi. L'uscita del circuito codificatore è una rappresentazione a 3 bit della posizione dell'interruttore. Molti interruttori comprendono codificatori in modo tale che le loro uscite sono codificate, manualmente come digit BCD (in logica negata).

Il codificatore produce uscite attive basse, così, per esempio, la posizione 5 dell'interruttore, che è collegata all'ingresso 2, produce un'uscita 2 in logica negata (o 5 in logica positiva). Potete verificare voi stessi la doppia negazione.

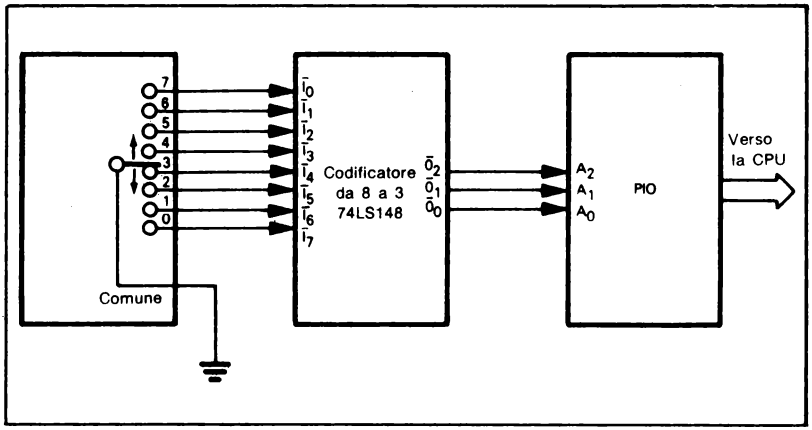
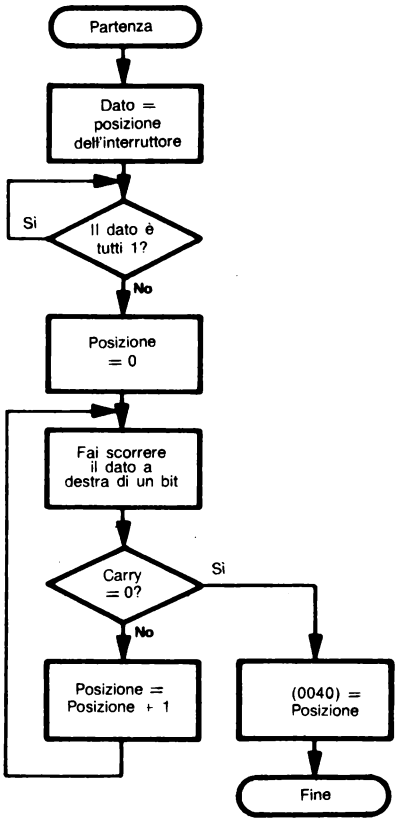


Fig. 11-14. Uno Switch a Posizioni Multiple con Codificatore

Diagramma di Flusso:



Programma Sorgente:

```
        LD      A,01001111B   ;RENDI LA PORTA AD UN INGRESSO
        OUT     (PIOCRA),A
CHKSW:  IN      A,(PIODRA)    ;PRENDI IL DATO DELL'INTERRUTTORE
        CP      0FFH          ;L'INTERRUTTORE È IN POSIZIONE?
        JR      Z,CHKSW       ;NO, ASPETTA CHE SIA IN POSIZIONE
        LD      B,0           ;POSIZIONE DELL'INTERRUTTORE = ZERO
CHPOS:  RRA                   ;IL BIT SUCCESSIVO È A TERRA?
        JR      NC,DONE       ;SÌ, SI È TROVATA LA POSIZIONE
                              ;   DELL'INTERRUTTORE
        INC     B             ;NO, INCREMENTA LA POSIZIONE
                              ;   DELL'INTERRUTTORE
        JR      CHPOS
DONE:   LD      HL,40H        ;MEMORIZZA LA POSIZIONE DELL'INTERRUTTORE
        LD      (HL),B
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Operativo)		
0000	3E		LD	A,01001111B
0001	4F			
0002	D3		OUT	(PIOCRA),A
0003	PIOCRA			
0004	DB	CHKSW:	IN	A,(PIODRA)
0005	PIODRA			
0006	FE		CP	0FFH
0007	FF			
0008	28		JR	Z,CHKSW
0009	FA			
000A	06		LD	B,0
000B	00			
000C	1F	CHPOS:	RRA	
000D	30		JR	NC,DONE
000E	03			
000F	04		INC	B
0010	18		JR	CHPOS
0011	FA			
0012	21	DONE:	LD	HL,40H
0013	40			
0014	00			
0015	70		LD	(HL),B
0016	76		HALT	

Si supponga che un interruttore imperfetto o un PIO difettoso facciano si che all'ingresso vi sia sempre $0FF_{16}$. Come potreste cambiare il programma in modo tale che esso possa rilevare questo errore?

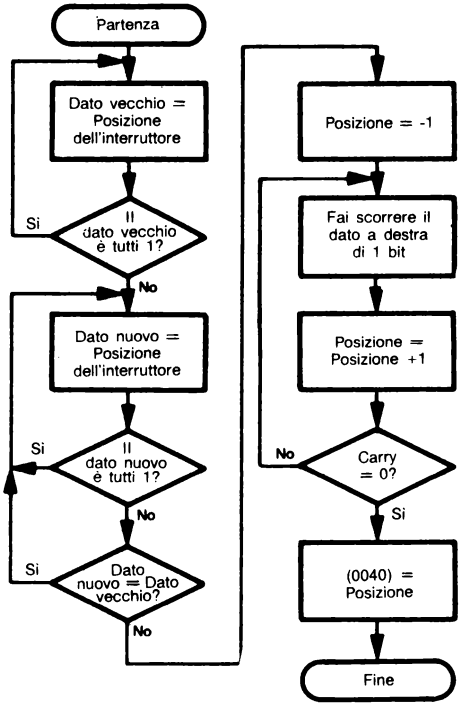
Nel programma sorgente c'è un salto incondizionato, JR CHPOS. Potete cambiare le condizioni iniziali in modo tale da rendere quest'istruzione non necessaria?

Quest'esempio ipotizza che i rimbalzi dell'interruttore siano annullati in hardware. Come dovreste cambiare il programma in modo da annullare i rimbalzi dell'interruttore in software?

Compito 2: Attesa per il cambiamento di posizione dell'interruttore

Scopo: Il programma attende che la posizione dell'interruttore cambi e sistema la nuova posizione (decodificata) nella locazione di memoria 0040. Il programma attende sino a che l'interruttore raggiunge la sua nuova posizione.

Diagramma di Flusso:



Programma Sorgente:

```
        LD      A,01001111B  ;RENDI LA PORTA A UN INGRESSO
        OUT     (PIOCRA),A
CHFST:  IN      A,(PIODRA)   ;PRENDI IL DATO DELL'INTERRUTTORE
        CP      0FFH         ;L'INTERRUTTORE È IN POSIZIONE?
        JR      Z,CHFST      ;NO, ASPETTA CHE SIA IN POSIZIONE
        LD      B,A
CHSEC:  IN      A,(PIODRA)   ;PRENDI IL NUOVO DATO DELL'INTERRUTTORE?
        CP      0FFH         ;L'INTERRUTTORE È IN POSIZIONE?
        JR      Z,CHSEC      ;NO, ASPETTA CHE SIA IN POSIZIONE
        CP      B            ;LA POSIZIONE È LA STESSA DI PRIMA?
        JR      Z,CHSEC      ;SÌ, ASPETTA CHE CAMBI
        LD      B,0FFH       ;POSIZIONE DELL'INTERRUTTORE = – 1
CHPOS:  INC     B            ;INCREMENTA LA POSIZIONE DELL'INTERRUTTORE
        RRA                  ;IL BIT SUCCESSIVO È A TERRA?
        JR      C,CHPOS      ;NO, CONTINUA A CERCARE LA POSIZIONE DI TERRA
        LD      HL,40H       ;MEMORIZZA LA POSIZIONE DELL'INTERRUTTORE
        LD      (HL),B
        HALT
```

Programma oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Operativo)		
0000	3E		LD	A,01001111B
0001	4F			
0002	D3		OUT	(PIOCRA),A
0003	PIOCRA			
0004	DB	CHFST:	IN	A,(PIODRA)
0005	PIODRA			
0006	FE		CP	0FFH
0007	FF			
0008	28		JR	Z,CHFST
0009	FA			
000A	47		LD	B,A
000B	DB	CHSEC:	IN	A,(PIODRA)
000C	PIODRA			
000D	FE		CP	0FFH
000E	FF			
000F	28		JR	Z,CHSEC
0010	FA			
0011	B8		CP	B
0012	28		JR	Z,CHSEC
0013	F7			
0014	06		LD	B,0FFH
0015	FF			
0016	04	CHPOS:	INC	B
0017	1F		RRA	
0018	38		JR	C,CHPOS
0019	FC			
001A	21		LD	HL,40H
001B	40			
001C	00			
001D	70		LD	(HL),B
001E	76		HALT	

Un metodo alternativo per determinare se l'interruttore è in una posizione è:

```
CHKSW:  IN      A,(PIODRA)
        INC     A
        JR      Z,CHKSW
```

Perchè questo lavoro? Cosa succede al dato di ingresso?

Un singolo LED

Scopo: Interfacciare un singolo diodo emettitore di luce con un microprocessore Z80. Il LED può essere collegato in modo tale che alla sua accensione corrisponda o uno zero logico o un uno logico.

Schema del circuito:

CONTROLLO DEL LED

La Figura 11-15 mostra la circuiteria richiesta per interfacciare un LED. Il LED si accende quando il suo anodo è positivo rispetto al suo catodo (Figura 11-15a). Perciò, potete o accendere il LED mettendo a massa il catodo mentre il computer fornisce un uno all'anodo (Figura 11-15b) o connettendo l'anodo a +5 volt mentre il computer dà al catodo uno zero (Figura 11-15c). L'impiego del catodo è l'approcio più comune. Il LED è più brillante quando funziona con correnti pulsanti di circa 10 o 50 mA applicate alcune centinaia di volte al secondo. I LED hanno un tempo di accensione molto breve (nel campo del microsecondo), così che essi bene si adattano al multiplexing (facendone funzionare parecchi con una sola porta). I circuiti a LED di solito necessitano di periferiche o di driver a transistor e resistenze limitatrici di corrente. I dispositivi MOS normalmente non possono pilotare direttamente i LED e farli brillare abbastanza per una facile visione.

Nota: Il PIO ha un latch d'uscita su ciascuna porta. Tuttavia, la porta B è normalmente impiegata come uscita, poiché essa ha capacità di pilotaggio alquanto maggiore. In particolare, le uscite della porta B sono in grado di pilotare transistor Darlington (fornendo un minimo di 1,5 mA a 1,5 V). I transistor Darlington sono transistor ad alto guadagno in grado di commutare grandi quantità di corrente ad alta velocità; essi sono molto utili nel pilotare solenoidi, relé, ed in altri dispositivi.

Compito: Accendere o spegnere la luce

Scopo: Il programma accende o spegne un singolo LED A. Si trasmette in uno logico al LED (si accende un display positivo o si spegne un display negativo).

Programma Sorgente:

(formazione iniziale del dato)

```
LD      A,00001111B   ;RENDI LA PORTA B UN'USCITA
OUT     (PIOCRB),A
LD      A,MASKP       ;PRENDI IL DATO PER IL LED
OUT     (PIODRB),A    ;INVIA IL DATO AL LED
HALT
```

Un impiego alternativo del modo di controllo è:

```
LD      A,11001111B   ;RENDI LA PORTA B UN CONTROLLO
OUT     (PIOCRB),A
SUB     A             ;RENDI TUTTE LE LINEE B USCITE
OUT     (PIOCRB),A
LD      A,MASKP       ;PRENDI IL DATO PER IL LED
OUT     (PIODRB),A    ;INVIA IL DATO AL LED
HALT
```

(aggiornamento del dato)

```
IN      A,(PIODRB)    ;PRENDI IL DATO VECCHIO
SET     LED,A         ;ACCENDI IL LED
OUT     (PIODRB),A    ;INVIA IL DATO AL LED
HALT
```

MASKP ha un bit uno nella posizione del LED e zeri in ogni altra parte. Si noti che possiamo leggere il Registro d'Uscita del dato del PIO quando il PIO è nel modo di uscita. Possiamo anche leggere qualunque combinazione di dati di ingresso e di dati del registro d'uscita quando il PIO è nel modo di controllo; la combinazione è definita dall'assegnazione degli ingressi e delle uscite.

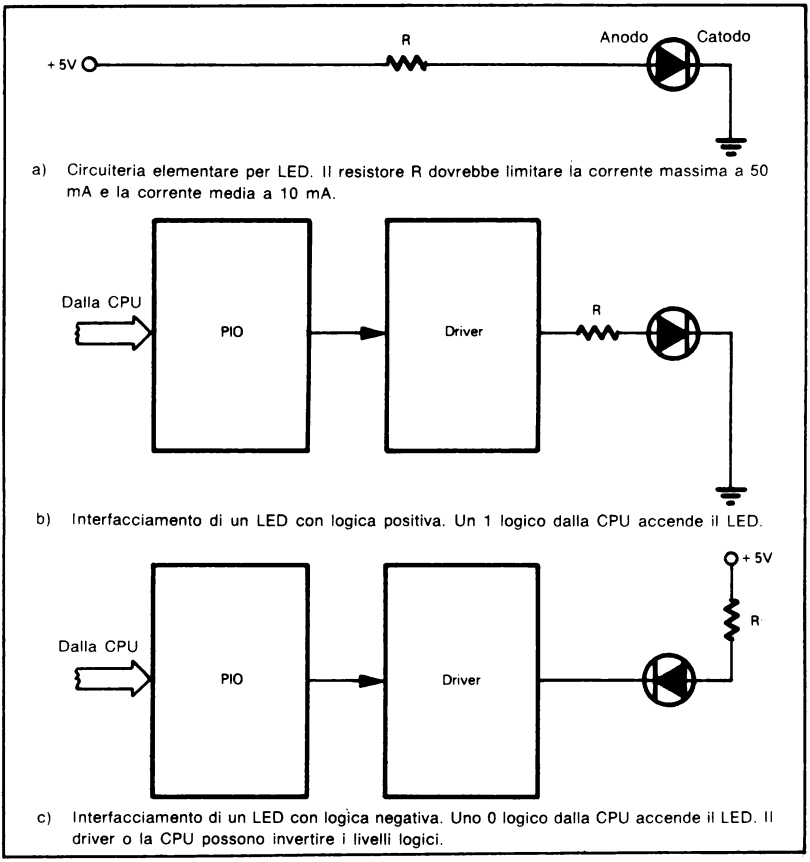


Figura 11-15. Realizzazione dell'Interfaccia di un LED

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Operativo)	
	(forma inizialmente il dato)		
0000	3E	LD	A,00001111B
0001	0F		
0002	D3	OUT	(PIOCRB),A
0003	PIOCRB		
0004	3E	LD	A,MASKP
0005	MASKP		
0006	D3	OUT	(PIODRB),A
0007	PIODRB		
0008	76	HALT	
	(aggiorna il dato)		
0009	DB	IN	A,(PIODRB)
000A	PIODRB		
000B	CB	SET	LED,A
000C	LED		
000D	D3	OUT	(PIODRB),A
000E	PIODRB		
000F	76	HALT	

B. Si trasmette uno zero logico al LED (si spegne un display positivo o si accende un display negativo).

Le differenze sono che MASKP deve essere sostituita dal suo complemento logico MASKN e SET LED,A deve essere sostituito da RES LED,A. Si noti che il secondo byte del codice oggetto per SET LED,A e per RES LED,A dipende dalla reale posizione del bit cui si riferisce il nome LED.

MASKN ha un bit zero nella posizione LED e degli uno nelle altre posizioni.

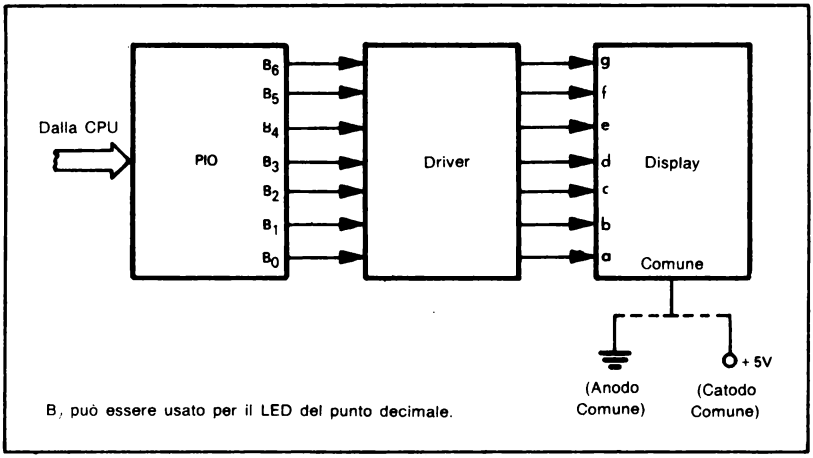


Figura 11-16. Realizzazione dell'Interfaccia di un Display a 7 Segmenti

Display LED a sette segmenti

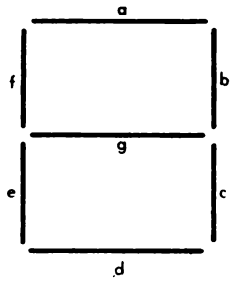
Scopo: Interfacciare un display LED a sette segmenti con un microprocessore Z80. Il display deve essere o ad anodo comune (logica negativa) o ad catodo comune (logica positiva).

Schema del circuito:

DISPLAY AD ANODO COMUNE O A CATODO COMUNE

La Figura 11-16 mostra la circuiteria richiesta per interfacciare un display a sette segmenti. Ogni segmento può avere uno, due, o più LED collegati nello stesso modo. Vi sono due modi per connettere i display. Uno consiste nel collegare tutti i catodi insieme a massa (vedi Figura 11-17a): questo è un display «a catodo comune». Collegando tutti gli anodi insieme a una tensione di alimentazione positiva (vedi Figura 11-17b) si realizza un display «ad anodo comune», ed uno zero logico ad un catodo accende un segmento. Così il display a catodo comune impiega logica positiva ed il display ad anodo comune impiega logica negativa. Entrambi i display richiedono driver e resistenze appropriate.

La linea Comune dal display è collegata o a massa o a +5 volt. I segmenti del display sono abitualmente denominati:



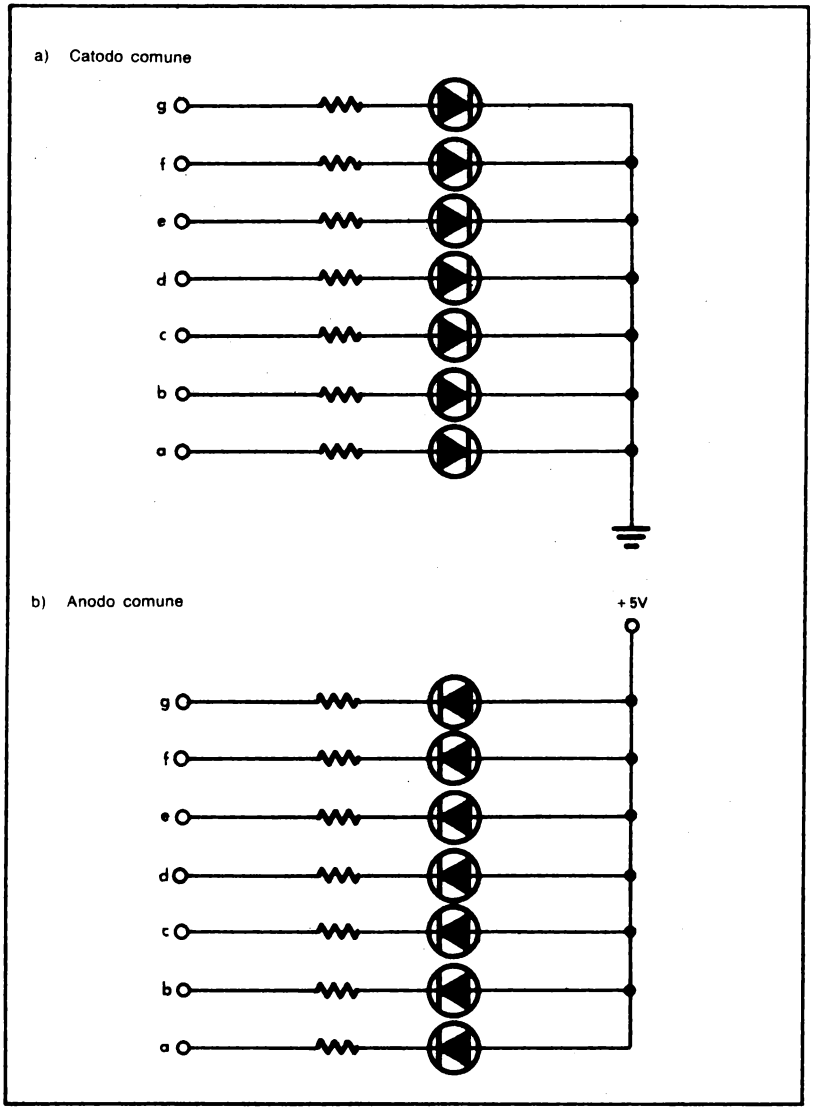


Figura 11-17. Disposizione del Display a 7 Segmenti.

Tabella 11-4. Rappresentazione su 7 Segmenti di numeri decimali

Numero	Rappresentazione esadecimale	
	Catodo Comune	Anodo comune
0	3F	40
1	06	79
2	5B	24
3	4F	30
4	66	19
5	6D	12
6	7D	02
7	07	78
8	7F	00
9	67	18
Il Bit 7 è sempre zero e gli altri sono g, f, e, d, c, b, a in ordine decrescente il significato.		

RAPPRESENTAZIONI A SETTE SEGMENTI

Nota: Il display a sette segmenti è molto impiegato poiché contiene il minor numero di segmenti controllati separatamente che possono fornire una rappresentazione riconoscibile di tutte le cifre decimali (si veda Figura 11-18 e la Tabella 11-4). I display a sette segmenti possono anche realizzare alcune lettere ed altri caratteri (si veda la Tabella 11-5). Migliori rappresentazioni richiedono un numero sostanzialmente più grande di segmenti e più circuiti[5]. Poiché i display a sette segmenti sono così comuni, decodificatori/driver per i sette segmenti a basso costo sono diventati largamente disponibili. I dispositivi più popolari sono il driver 7447 ad anodo comune ed il driver 7448 a catodo comune[6]; questi dispositivi hanno ingressi di Lamp Test (che accendono tutti i segmenti) e ingressi e uscite di spegnimento (per la soppressione degli zeri non significativi di testa e di coda).

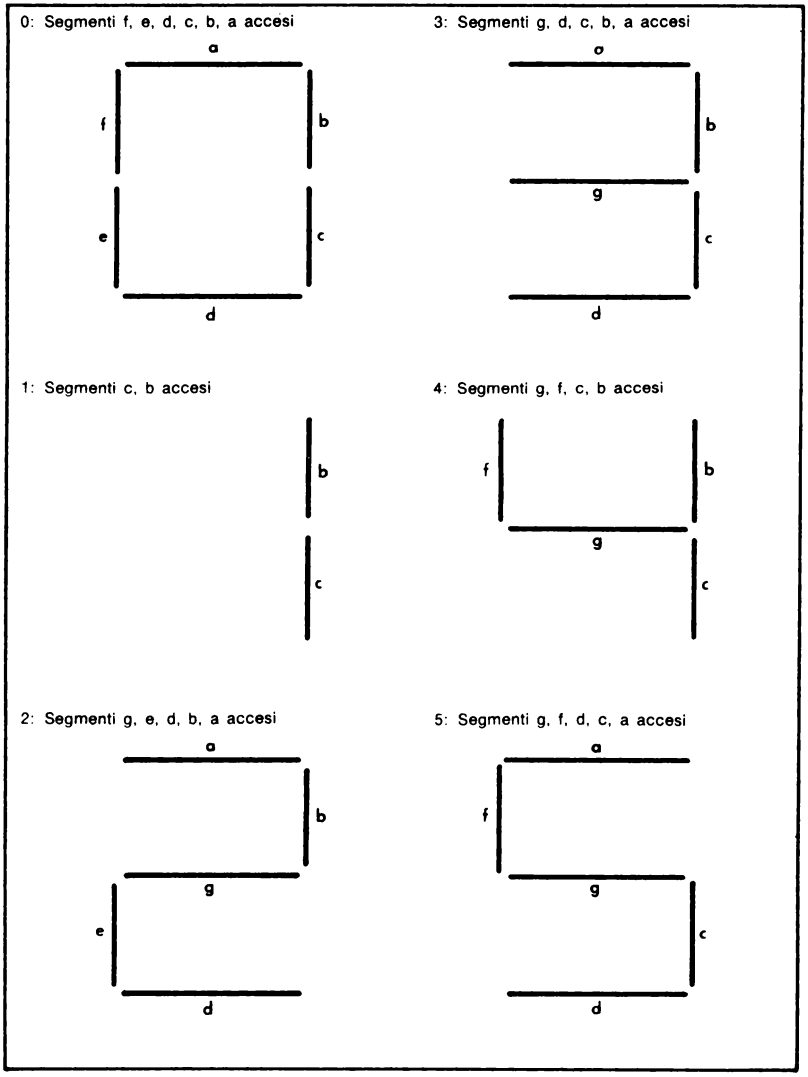
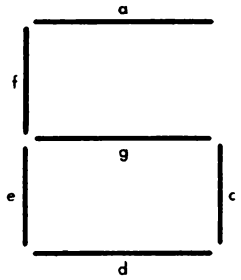


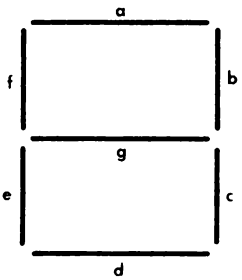
Figura 11-18. Rappresentazione a 7 Segmenti delle Cifre Decimali

6: Segmenti g, f, e, d, c, a accesi



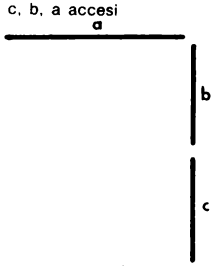
Si noti che la rappresentazione alternativa con a spento può essere riservata per 'b' minuscolo.

8: Segmenti g, f, e, d, c, b, a accesi

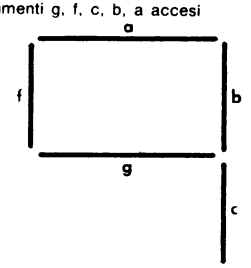


Questo è lo stesso di LAMP TEST.

7: Segmenti c, b, a accesi



9: Segmenti g, f, c, b, a accesi



Un'alternativa ha acceso pure il segmento d.

Figure 11-18. Rappresentazione a 7 Segmenti delle Cifre Decimali

Tabella 11-5. Rappresentazioni su 7 Segmenti di lettere e simboli

Lettere Maiuscole

Lettera	Rappresentazione Esadecimale	
	Catodo Comune	Anodo Comune
A	77	08
C	39	46
E	79	06
F	71	0E
H	76	09
I	06	79
J	1E	61
L	38	47
O	3F	40
P	73	0C
U	3E	41
Y	66	19

Lettere Minuscole e Caratteri Speciali

Carattere	Rappresentazione Esadecimale	
	Catodo Comune	Anodo Comune
b	7C	03
c	58	27
d	5E	21
h	74	0B
n	54	2B
o	5C	23
r	50	2F
u	1C	63
-	40	3F
?	53	2C

Compito 1: Visualizzare una cifra decimale

Scopo: Visualizzare il contenuto della locazionie di memoria 0040 su un dispositivo a sette segmenti se essa contiene una cifra decimale. Altrimenti, spegnere il display.

Problemi Campione:

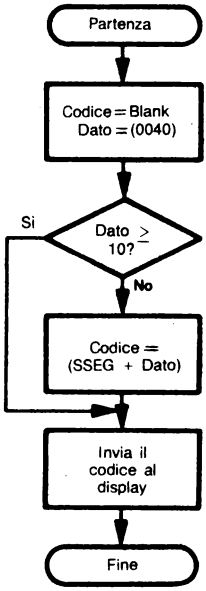
a. (0040) = 05

Il risultato è 5 nel display

b. (0040) = 66

Il risultato è un display spento

Diagramma di flusso:



Programma Sorgente:

```
        LD      A,00001111B  ;RENDI LA PORTA B UN'USCITA
        OUT     (PIOCRB),A
        LD      B,BLANK      ;PRENDI IL CODICE DI BLANK
        LD      A,(40H)      ;PRENDI IL DATO
        CP      10           ;IL DATO È UN DIGIT DECIMALE?
        JR      NC,DSPLY     ;NO, SPEGNI IL DISPLAY
        LD      DE,SSEG      ;PRENDI L'INDIRIZZO DI BASE DELLA TABELLA
                             ;   DEI SETTE SEGMENTI
        LD      H,0          ;TRASFORMA IL DATO IN UN INDICE A 16 BIT
        LD      L,A
        ADD     HL,DE        ;ACCEDI ALL'ELEMENTO DELLA TABELLA
        LD      B,(HL)       ;PRENDI IL CODICE A SETTE SEGMENTI
DSPLY   LD      A,B
        OUT     (PIODRB),A   ;INVIA IL CODICE AL DISPLAY
        HALT
```

BLANK è 00 per un display a catodo comune, è FF per un display ad anodo comune. Un modo di procedere alternativo potrebbe essere mettere il codice blank alla fine della tabella e sostituire tutti i valori impropri del dato con 10, cioè,

```
        LD      A,(40H)      ;PRENDI IL DATO
        CP      10           ;IL DATO È UN DIGIT DECIMALE?
        JR      C,CNVRT      ;SÌ, CONVERTILO DIRETTAMENTE IN CODICE
                             ;   A SETTE SEGMENTI
        LD      A,10         ;NO, PRENDI L'INDICE PER IL CODICE DI BLANK
CNVRT   LD      DE,SSEG      ;PRENDI L'INDIRIZZO DI BASE DELLA TABELLA
                             ;   A SETTE SEGMENTI
```

La tabella SSEG è la rappresentazione delle cifre decimali della Tabella 11-4 o secondo la configurazione a catodo comune o secondo quella ad anodo comune.

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)		Istruzione (Codice Operativo)	
0000	3E		LD	A,00001111B
0001	0F			
0002	D3		OUT	(PIOCRB),A
0003	PIOCRB			
0004	06		LD	B,BLANK
0005	BLANK			
0006	3A		LD	A,(40H)
0007	40			
0008	00			
0009	FE		CP	10
000A	0A			
000B	30		JR	NC,DSPLY
000C	08			
000D	11		LD	DE,SSEG
000E	20			
000F	00			
0010	26		LD	H,0
0011	00			
0012	6F		LD	L,A
0013	19		ADD	HL,DE
0014	46		LD	B,(HL)
0015	78	DSPLY:	LD	A,B
0016	D3		OUT	(PIODRB),A
0017	PIODRB			
0018	76		HALT	
0020-0029		SSEG:	(tabella dei codici a sette segmenti)	

Parecchi display devono essere multiplexed, come mostrato in Figura 11-19. Un breve strobe sulla linea BRDY fornisce il clock al contatore e dirige il dato al display successivo. Si noti che BRDY è collegato direttamente indietro a $\overline{B\ STB}$, cioè la linea pronta fornisce essenzialmente il proprio riconoscimento. La temporizzazione del PIO è tale che questa connessione ha come effetto uno strobe con una durata di un periodo di clock. Uno strobe così breve è esattamente ciò che il contatore richiede. Il RESET fa partire il contatore decimale a nove in modo tale che la prima operazione di uscita azzera il contatore e dirige il dato al primo display.

Il programma seguente impiega la routine di ritardo per inviare a ciascuno dei dieci display a catodo comune un impulso di 1 ms.

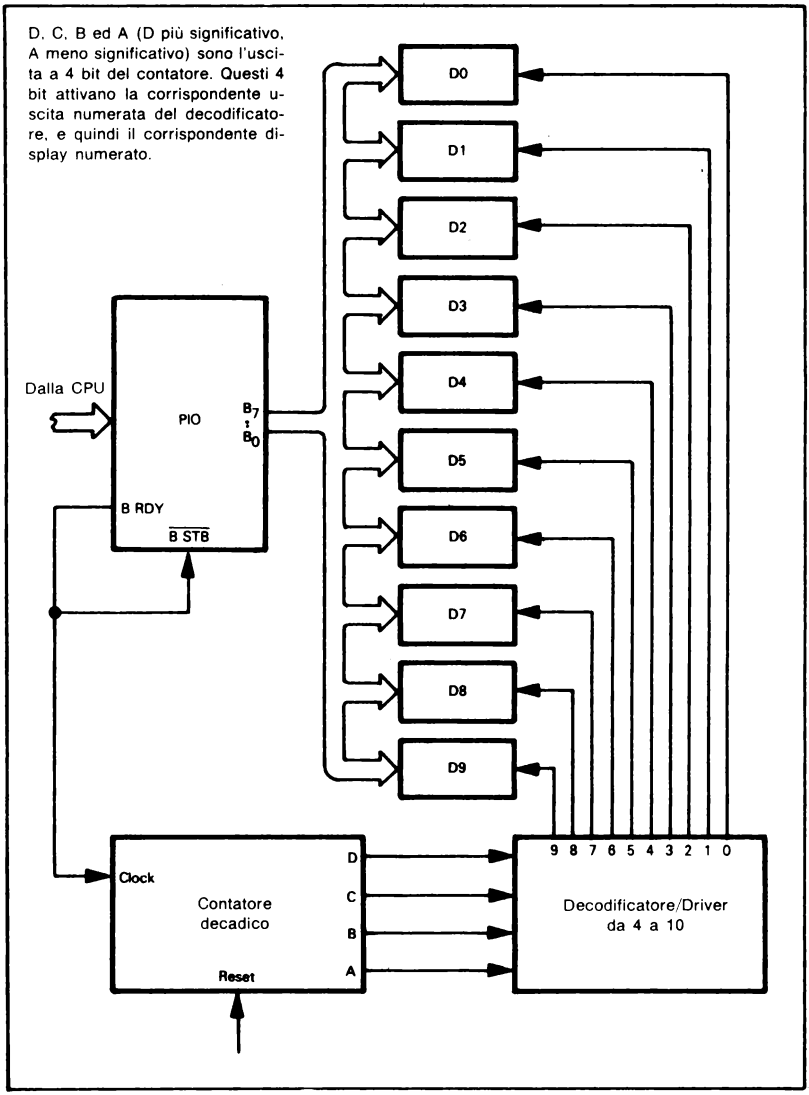


Figura 11-19. Display a 7 Segmenti Multiplexed

Compito 2: Visualizzare dieci cifre decimali.

Scopo: Visualizzare i contenuti delle locazioni di memoria da 0040 a 0049 su dieci display a sette segmenti che sono multiplexed con un contatore ed un decodificatore.

Problema Campione:

(0040) = 66
(0041) = 3F
(0042) = 7F
(0043) = 7F
(0044) = 06
(0045) = 5B
(0046) = 07
(0047) = 4F
(0048) = 6D
(0049) = 7D

Letture dei display 4088127356

Programma Sorgente:

```
        LD      A,00001111B  ;RENDI LA PORTA B UN'USCITA
        OUT     (PIOCRB),A
DRUN:   LD      HL,40H       ;PUNTA ALL'INIZIO DEI DATI
        LD      B,10         ;NUMERO DEI DISPLAY = 10
        LD      C,PIODRB     ;PRENDI IL NUMERO DELLA PORTA
DSPLY:  OUT1                 ;INVIA IL DATO AL DISPLAY
        CALL    DELAY        ;ASPETTA 1 MS
        JR      NZ,DSPLY     ;CONTA 1 DISPLAY
        JR      DRUN         ;INIZIA UN'ALTRA SCANSIONE
```

Qui dobbiamo selezionare il modo d'uscita del PIO, poiché il circuito impiega i segnali di handshake.

Si noti che OUTI trasmette il dato alla porta d'uscita indirizzata dal registro C, incrementa l'indirizzo nella coppia di Registri HL, e decrementa il contatore nel Registro B. Abbiamo presupposto che la subroutine DELAY non influenzi il flag Z cosicché esso possa essere impiegato in seguito per un salto condizionato.

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)		Istruzione (Codice Operativo)	
0000	3E		LD	A,00001111B
0001	0F			
0002	D3		OUT	(PIOCRB),A
0003	PIOCRB			
0004	21	DRUN:	LD	HL,40H
0005	40			
0006	00			
0007	06		LD	B,10
0008	0A			
0009	0E		LD	C,PIODRB
000A	PIODRB			
000B	ED	DSPLY:	OUTI	
000C	A3			
000D	CD		CALL	DELAY
000E	30			
000F	00			
0010	20		JR	NZ,DSPLY
0011	F9			
0012	18		JR	DRUN
0013	F0			

PROBLEMI

1) Un Pulsante Aperto-Chiuso

Scopo: Ogni chiusura del pulsante fa il complemento (inverte) di tutti i bit della locazione di memoria 0040. La locazione inizialmente contiene zero. Il programma dovrebbe esaminare continuamente il pulsante e complementare la locazione 0040 per ciascuna chiusura. Potete desiderare di complementare, invece, la porta di uscita di un display, in modo da rendere i risultati più facili da vedere.

Caso campione:

La locazione 0040 inizialmente contiene zero.

La prima chiusura del pulsante cambia la locazione 0040 in FF (esadecimale), la seconda la cambia di nuovo in zero, la terza la riporta a FF (esadecimale), etc. Si assuma che il rimbalzo del pulsante sia annullato in hardware. Come prendere l'annullamento del rimbalzo nel vostro programma?

2) Annullamento del rimbalzo di un interruttore in software

Scopo: Annullamento del rimbalzo di un interruttore meccanico aspettando due letture, intercalate da un tempo di annullamento del rimbalzo per ottenere lo stesso risultato. Si assuma che il tempo dell'annullamento del rimbalzo (in ms) sia nella locazione di memoria 0040 e si posizioni l'interruttore nella locazione di memoria 0041.

Problema Campione:

(0040) = 03 fa sì che il programma attenda 3 ms tra due letture.

3) Controllo di un Interruttore a Rotazione

Scopo: Un altro interruttore funziona come un interruttore di Load (carico) per un interruttore a rotazione non codificato a 4 posizioni. La CPU attende che l'interruttore Load sia chiuso (sia zero) ed allora legge la posizione dell'interruttore a rotazione. Questa procedura consente all'operatore di spostare l'interruttore a rotazione nella sua posizione finale prima che la CPU cerchi di leggerla. Il programma dovrebbe sistemare la posizione dell'interruttore a rotazione nella locazione di memoria 0040. Il rimbalzo dell'interruttore Load è annullato in software.

Problema Campione:

Sistemare l'interruttore nella posizione 2. Chiudere l'interruttore Load.

Risultato: (0040) = 02

4) Luci di Segnalazione per le Posizioni degli Interruttori

Scopo: Un insieme di otto interruttori dovrà avere le proprie posizioni segnalate da otto LED. Questo vuol dire, che se l'interruttore è chiuso (zero), il LED sarà acceso, altrimenti il LED sarà spento. Si assume che la porta di uscita della CPU sia collegata ai catodi dei LED.

Problema Campione:

INTERRUTTORE	0	CHIUSO
INTERRUTTORE	1	APERTO
INTERRUTTORE	2	CHIUSO
INTERRUTTORE	3	APERTO
INTERRUTTORE	4	APERTO
INTERRUTTORE	5	CHIUSO
INTERRUTTORE	6	CHIUSO
INTERRUTTORE	7	APERTO

Risultato:

LED	0	ACCESO
LED	1	SPENTO
LED	2	ACCESO
LED	3	SPENTO
LED	4	SPENTO
LED	5	ACCESO
LED	6	ACCESO
LED	7	SPENTO

Come dovreste modificare il programma in modo tale che un interruttore collegato al bit 7 della Porta A del PIO#2 determini in ogni caso che i display siano attivi (cioè, se l'interruttore di controllo è chiuso, i display collegati alla Porta B rispecchiano i display collegati alla Porta A; se l'interruttore di controllo è aperto; i display siano sempre spenti)? Un interruttore di controllo è utile quando i display possono distrarre l'operatore, come in aeroplano.

Come dovreste modificare il programma in modo tale da rendere l'interruttore di controllo come un pulsante aperto-chiuso; cioè, ogni chiusura dà l'opposto del precedente stato dei display? Supponiamo che i display partano nello stato attivo e che il programma esamini ed annulli i rimbalzi del pulsante prima di trasmettere il dato ai display.

5) Contare su un Display a Sette Segmenti

Scopo: Il programma conterà da 0 a 9 continuamente su un display a sette segmenti, partendo da zero.

Suggerimento: Provare intervalli di tempo diversi per i display e vedere cosa succede. Quando il conteggio diventa visibile? Che cosa accade se il display viene spento per una parte del tempo?

DISPOSITIVI DI I/O PIÙ COMPLESSI

Dispositivi di I/O più complessi differiscono dalle semplici tastiere, interruttori, e display in ciò:

1) Trasferiscono dati a velocità più alta.
2) Possono avere i propri clock e le proprie temporizzazioni interne.
3) Producono informazioni di stato e richiedono informazioni di controllo, come pure il trasferimento di dati.

A causa delle loro alte velocità di dato non potete gestire questi dispositivi a caso. Se l'elaboratore non fornisce il servizio appropriato, il sistema può perdere dati di ingresso o produrre dati di uscita errati. State perciò lavorando in condizioni molto più stringenti di quelle concernenti dispositivi più semplici. Gli interrupt rappresentano un metodo conveniente per maneggiare dispositivi di I/O complessi, come vedremo nel capitolo 12.

SINCRONIZZAZIONE CON DISPOSITIVI DI I/O

Periferiche quali tastiere, telescriventi, cassette, e floppy disk producono al proprio interno la propria temporizzazione. Questi dispositivi forniscono correnti di dati, separati da intervalli di tempo specifici. Il computer deve sincronizzare l'operazione iniziale di ingresso o di uscita col clock della periferica e fornire quindi l'opportuno intervallo tra due successive operazioni. Un semplice ciclo di ritardo simile a quello mostrato precedentemente può produrre l'intervallo di tempo. La sincronizzazione può richiedere una sola o più delle seguenti procedure:

1) Ricerca di una transizione su una linea di clock o di strobe fornita dalla periferica ai fini della temporizzazione. Un semplice approccio potrebbe essere collegare lo strobe all'ingresso $\overline{STB}$ di un PIO e cercare una variazione nell'uscita di interrupt ($\overline{INT}$). Tuttavia, non c'è alcun modo per indirizzare direttamente l'uscita $\overline{INT}$ (e pertanto determinare il suo valore) e non c'è alcun modo per annullarla tranne che attraverso una routine di servizio dell'interrupt. In tale modo per usare il PIO in un sistema a «polling», si deve rendere lo strobe disponibile ad una porta di ingresso e metterlo in un latch, se necessario. Se lo strobe deve essere messo in un latch, si deve pure fornire un circuito per azzerare il latch come conseguenza di un successivo trasferimento di ingresso o di uscita.
2) Trovare il centro dell'intervallo di tempo durante il quale il dato è stabile. Sarebbe preferibile determinare il valore del dato nel centro dell'impulso piuttosto che alle estremità, dove il dato potrebbe essere in fase di cambiamento. Trovare il centro, richiede un ritardo di metà dell'intervallo di trasmissione (tempo di bit) dopo il fronte. Campionare il dato nel centro significa pure che piccoli errori di temporizzazione hanno piccoli effetti sulla precisione della ricezione.
3) Riconoscere uno speciale codice d'inizio. Ciò è facile se il codice è un singolo bit o se abbiamo qualche informazione della temporizzazione. Questa procedura è più complessa se il codice è lungo e potrebbe partire in un istante qualunque. Sarà necessario fare degli scorrimenti per determinare dove il trasmettitore ha fatto partire i suoi bit, i suoi caratteri o i suoi messaggi (ciò spesso è chiamata una ricerca per il «framing» corretto).
4) Campionare i dati parecchie volte. Questo riduce la probabilità di ricevere dati non correttamente da linee rumorose. Si può usare una logica maggioritaria (quale il migliore di 3 su 5 o di 5 su 8) per decidere su valore reale del dato.

La ricezione è, naturalmente, molto più difficile della trasmissione, poiché la periferica controlla la ricezione e il computer deve interpretare le informazioni di temporizzazione generate dalla periferica. In trasmissione, il computer fornisce la opportuna temporizzazione e l'opportuno formato per ogni specifica periferica.

INFORMAZIONI DI CONTROLLO E DI STATO

Le periferiche possono richiedere o fornire altre informazioni oltre i dati e la temporizzazione. Ci riferiamo alle altre informazioni trasmesse dal computer come alle «informazioni di controllo»: esse possono scegliere modi di funzionamento, far iniziare o fermare processi, generare clock per registri, abilitare buffer, scegliere formati o protocolli, fornire display per operatore, contare operazioni o identificare il tipo e la priorità dell'operazione. Facciamo riferimento alle altre informazioni trasmesse dalla periferica come «informazioni di stato»; esse possono indicare il modo di funzionamento, il fatto che il dispositivo è pronto, la presenza di condizioni di errore, il formato del protocollo usato ed altri stati o condizioni.

Il computer gestisce informazioni di controllo e di stato come dati. Queste informazioni cambiano raramente, anche se i dati reali possono essere trasferiti ad altre velocità. Le informazioni di controllo o di stato possono essere bit singoli, digit, parole o parole multiple. Spesso si combinano e si manipolano i bit singoli o brevi campi da una sola porta d'ingresso o d'uscita.

Combinando informazioni di stato e di controllo in byte si riduce il numero totale di indirizzi di porte di I/O richieste dalle periferiche. Tuttavia la combinazione significa che il bit d'ingresso di un singolo stato deve essere interpretato separatamente e che i bit d'uscita di controllo devono essere determinati separatamente. Le procedure per isolare bit di stato e mettere ad uno o a zero i bit di controllo sono le seguenti:

Separazione dei Bit di Stato

SEPARAZIONE DELLE INFORMAZIONI DI STATO

Passo 1) Leggere il dato di stato dalla periferica
Passo 2) AND logico con una maschera (la maschera ha degli uni nelle posizioni di bit che devono essere esaminate e zero altrove)
Passo 3) Spostare i bit separati nelle posizioni dei bit meno significativi.

Se il campo è composto da un singolo bit, il Passo 2 non è necessario poiché si può controllare il bit con l'istruzione BIT. Se il bit singolo è il più significativo, vicino al più significativo o nella posizione meno significativa, si possono usare istruzioni logiche di scorrimento (AND A o OR A) per determinare il suo valore. Ricordate pure che le istruzioni di ingresso con indirizzamento indiretto, tramite registro (per esempio IN A, (C)) influenzano il flag di Segno. Queste posizioni di bit alquanto più accessibili sono spesso riservate per le informazioni di stato usate più frequentemente. Dovreste provare a scrivere le sequenze d'istruzioni richieste per l'elaboratore Z80.

Il Passo 3 non è necessario se il campo è composto da un singolo bit, poiché il flag Zero conterrà il complemento di quel bit dopo il Passo 2 (verificatelo!). Un'istruzione di Shift o di Load può sostituire il Passo 2 se il campo è composto da un singolo bit e se occupa la posizione del bit meno significativo, più significativo o vicino al più significativo. Queste posizioni sono spesso riservate per le informazioni di stato usate più frequentemente. Dovreste provare a scrivere le sequenze di istruzioni richieste per l'elaboratore 6800.

Posizionamento ed Azzeramento dei Bit di Controllo

COMBINAZIONE DELLE INFORMAZIONI DI CONTROLLO

Passo 1) Leggere le precedenti informazioni di controllo
Passo 2) AND logico con una maschera per azzerare i bit (la maschera ha degli zero nelle posizioni dei bit da azzerare, degli uno altrove) da azzerare, degli uno altrove)
Passo 3) OR logico con una maschera per posizionare ad 1 i bit (la maschera ha degli 1 nelle posizioni dei bit da posizionare, degli zeri altrove)
Passo 4) Inviare nuove informazioni di controllo alla periferica

Ecco che di nuovo la procedura è più semplice se il campo è composto da un bit singolo e se occupa una posizione alla fine della parola.

Ecco alcuni esempi di separazione e di combinazione di bit:

1) Un campo a 3 bit nelle posizioni da 2 a 4 di registro di dato di un PIO è un fattore scalare. Porre quel fattore nell'Accumulatore.

```
;
;LEGGI IL DATO DI STATO DALLA PORTA DI INGRESSO
;
        IN      A,(PIODR)       ;LEGGI IL DATO DI STATO
;
;MASCHERA IL FATTORE DI PESO E FAI UNO SCORRIMENTO
;
        AND     00011100B       ;MASCHERA IL FATTORE DI PESO
        RRCA                    ;FAI SCORRERE DUE VOLTE PER NORMALIZZARE
        RRCA
```

2) L'Accumulatore contiene un campo a 2 bit che devono essere messi nelle posizioni 3 a 4 del registro dati di un PIO.

```
;
;SPOSTA IL DATO VERSO LE POSIZIONI DEL CAMPO
;
        RLA                     ;FAI SCORRERE IL DATO
                                ;   NELLE POSIZIONI 3 E 4 DEI BIT
        RLA
        RLA
        AND     00011000B       ;AZZERA GLI ALTRI BIT
        LD      B,A             ;SALVA IL VALORE DEL NUOVO CAMPO
;
;COMBINA LE NUOVE POSIZIONI DI CAMPO CON IL DATO VECCHIO
;
        IN      A,(PIODR)       ;AZZERA IL VALORE DEL VECCHIO CAMPO
        AND     11100111B       ;INSERISCI IL VALORE DEL NUOVO CAMPO
        OUT     (PIODR),A
```

DOCUMENTAZIONE DEI TRASFERIMENTI DI STATO E DI CONTROLLO

La documentazione è un problema serio nella gestione delle informazioni di controllo e di stato. I significati degli ingressi di stato o delle uscite di controllo raramente sono ovvi. Il programmatore dovrebbe indicare chiaramente gli scopi delle operazioni di ingresso e di uscita nei commenti, per esempio «CONTROLLO SE IL LETTORE È ON», «SCEGLIERE PURE L'OPZIONE DI PARITÀ», o «ATTIVARE IL CONTATORE DI VELOCITÀ DEI BIT». La manipolazione dei bit, le istruzioni Logiche e di Shift diventerebbero altrimenti molto difficili da ricordare, da capire per il debugging.

Tabella 11-6. Confronto fra collegamenti singoli e collegamenti a Matrice per Tastiera

Dimensione della tastiera	Numero di Linee con Collegamenti Indipendenti	Numero di Linee con Collegamenti a Matrice
3 x 3	9	6
4 x 4	16	8
4 x 6	24	10
5 x 5	25	10
6 x 6	36	12
6 x 8	48	14
8 x 8	64	16

ESEMPI

Una Tastiera Non codificata

Scopo: Riconoscere la chiusura di un tasto di una tastiera 3 x 3 non codificata e porre il numero del tasto premuto nell'Accumulatore.

Le tastiere sono degli insiemi di interruttori (vedere la Figura 11-20). Numeri piccoli di tasti sono più facili da gestire se ogni tasto è collegato separatamente ad un bit di una porta d'ingresso. Interfacciare le tastiere equivale ad interfacciare un insieme di interruttori.

TASTIERA A MATRICE

Le tastiere con più di otto tasti richiedono più di una porta d'ingresso e perciò operazioni con multibyte. Questo crea una perdita di tempo particolarmente se i tasti sono separati logicamente, come in un computer o in una tastiera di terminale dove l'utilizzatore ne batte uno per volta. Il numero delle linee di ingresso può essere ridotto collegando i tasti a matrice, come mostrato in Figura 11-21. Ora ogni tasto rappresenta un potenziale collegamento tra una riga ed una colonna. La matrice della tastiera richiede n + m linee esterne, dove n è il numero di righe ed m è il numero di colonne. Questo in confronto alle n x m linee esterne se ogni tasto è separato. La Tabella 11-6 confronta il numero di tasti richiesto da configurazioni tipiche.

SCANSIONE DELLA TASTIERA

Un programma può determinare quale tasto è stato premuto usando le linee esterne della matrice. La procedura d'uso è la «scansione della tastiera». Mettiamo a terra la Riga 0 ed esaminiamo le linee di colonna. Se qualche linea è a terra, in qualche riga è stato premuto un tasto, provocando un collegamento riga con colonna. Possiamo determinare quale tasto è stato premuto determinando quale linea di colonna è a terra; cioè quale bit della porta d'ingresso è a zero. Se nessuna linea di colonna è a terra, procediamo con la Riga 1 e ripetiamo la scansione. È da notare che possiamo controllare se non è stato premuto nessun tasto mettendo a terra tutte le righe contemporaneamente ed esaminando le colonne.

La scansione della tastiera richiede che le linee di riga siano collegate ad una porta di uscita e che le linee di colonna siano collegate ad una porta di ingresso. La Figura 11-22 mostra l'adattamento. La CPU può mettere a terra una riga particolarmente ponendo uno zero nel bit opportuno della porta di uscita e degli uni negli altri bit.

La CPU può determinare lo stato di una colonna particolare esaminando il bit opportuno della porta d'ingresso.

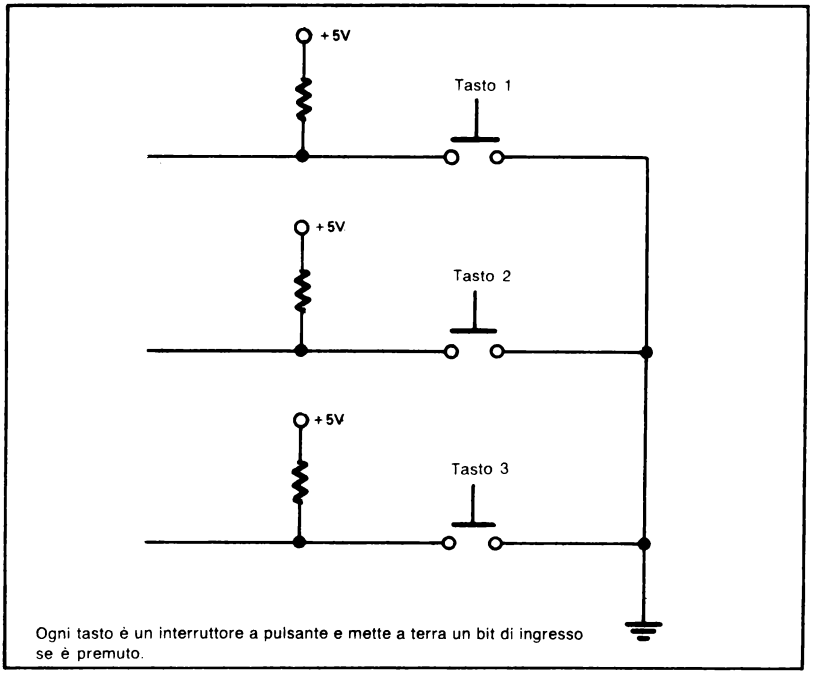


Figura 11-20. Una Piccola Tastiera

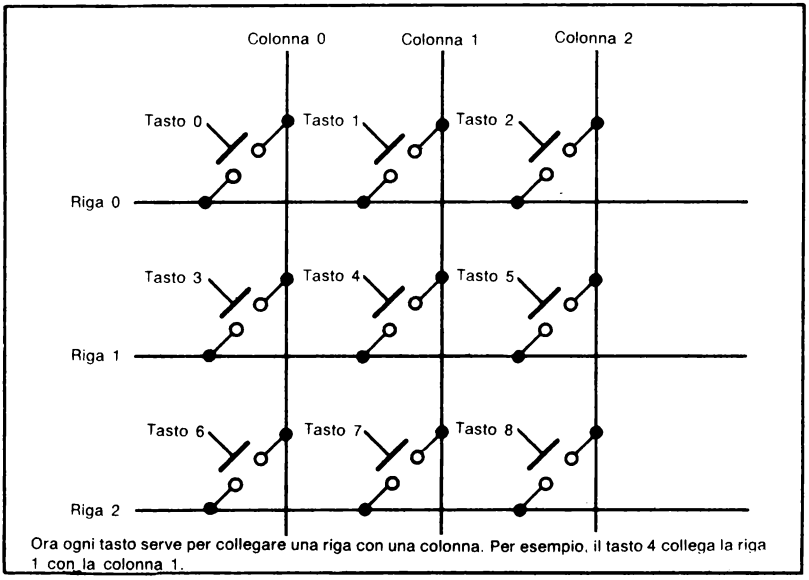


Figura 11-21. Una Tastiera a Matrice

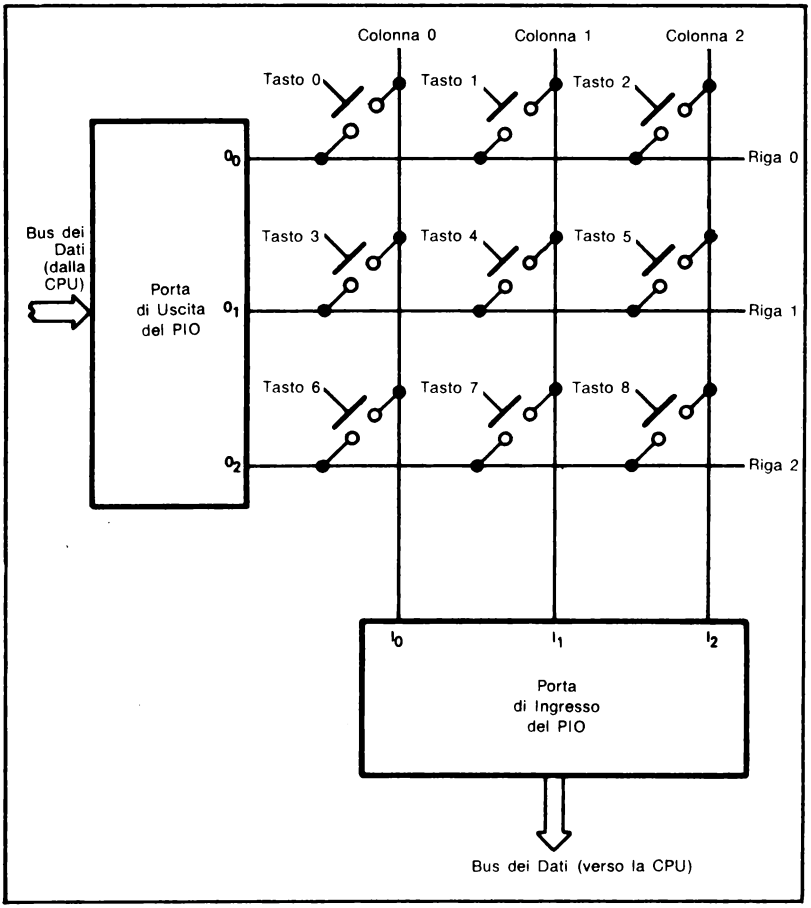


Figura 11-22. Disposizione I/O per l'Esplorazione di una Tastiera

Compito 1: Determinare la chiusura di un tasto.

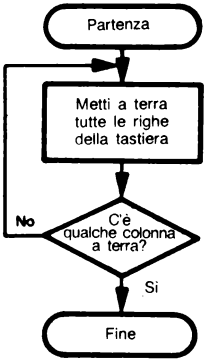
ATTESA DELLA CHIUSURA DI UN TASTO

Scopo: Attendere la pressione di un tasto.

La procedura è la seguente:

1) Mettere a terra tutte le righe azzerando tutti i bit di uscita.
2) Andare a prendere gli ingressi delle colonne leggendo la porta d'ingresso.
3) Tornare al Passo 1 se tutti gli ingressi delle colonne sono ad uno.

Diagramma di Flusso:



Programma Sorgente:

```
        LD      A,01001111B   ;RENDI LA PORTA A UN INGRESSO
        OUT     (PIOCRA),A
        LD      A,00001111B   ;RENDI LA PORTA B UN'USCITA
        OUT     (PIOCRB),A
        SUB     A             ;METTI A TERRA TUTTE LE RIGHE DELLA TASTIERA
        OUT     (PIODRB),A
WAITK:  IN      A,(PIODRA)    ;PRENDI IL DATO DI COLONNA DELLA TASTIERA
        AND     00000111B     ;MASCHERA I BIT DELLA COLONNA
        CP      00000111B     ;C'È QUALCHE COLONNA A TERRA?
        JR      Z,WAITK       ;NO, ASPETTA FINO A CHE NON LO SIA
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Operativo)		
0000	3E		LD	A,01001111B
0001	4F			
0002	D3		OUT	(PIOCRA),A
0003	PIOCRA			
0004	3E		LD	A,00001111B
0005	0F			
0006	D3		OUT	(PIOCRB),A
0007	PIOCRB			
0008	97		SUB	A
0009	D3		OUT	(PIODRB),A
000A	PIODRB			
000B	DB	WAITK:	IN	A,(PIODRA)
000C	PIODRA			
000D	E6		AND	00000111B
000E	07			
000F	FE		CP	00000111B
0010	07			
0011	28		JR	Z,WAITK
0012	F8			
0013	76		HALT	

La Porta B del PIO è la porta d'uscita della tastiera e la Porta A è la porta d'ingresso.

La mascheratura dei bit di colonna elimina ogni problema che potrebbe essere provocato dagli stati delle linee d'ingresso non utilizzate.

Potremmo generalizzare la routine dando un nome alle uscite ed alle combinazioni di mascheratura:

```
ALLG    EQU     11111000B
OPEN    EQU     00000111B
```

Questi nomi potrebbero essere usati nel programma reale; una tastiera diversa potrebbe richiedere solo una variazione nelle definizioni ed un riassemblaggio.

Naturalmente una sola porta del PIO è tutto ciò che è realmente necessario per una tastiera 3 x 3 o 4 x 4. Provate a riscrivere il programma in modo da usare solo la Porta A. Il PIO deve essere posizionato nel modo di controllo in modo che le linee possano essere selezionate una per una come ingressi o come uscite.

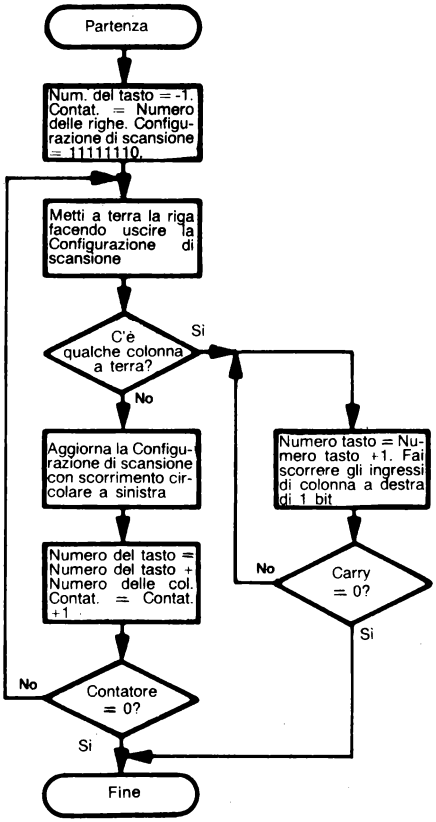
Compito 2: Identificare il tasto

Scopo: Identificare la chiusura di un tasto ponendo il numero del tasto nell'Accumulatore.

La procedura è la seguente:

1) Posizionare a -1 il numero del tasto, a tutti uno tranne che il bit 0, che va messo a zero, i bit del contatore del numero di righe e della combinazione d'uscita.
2) Mettere a terra una riga inviando la combinazione di uscita alla porta d'uscita della tastiera.
3) Aggiornare la combinazione di uscita facendo scorrere il bit zero a sinistra di una posizione.
4) Andare a prendere gli ingressi delle colonne leggendo la porta d'ingresso.
5) Se qualche ingresso di colonna è a zero, passare al Passo 8.
6) Sommare il numero delle colonne al numero del tasto per raggiungere la riga successiva.
7) Decrementare il contatore. Andare al Passo 2 se non è stata esplorata qualche riga, altrimenti andare al Passo 10.
8) Sommare 1 al numero del tasto. Fare scorrere gli ingressi della colonna a destra di un bit.
9) Se il Carry = 1, tornare al Passo 8.
10) Fine del programma.

Diagramma di Flusso:



Programma Sorgente:

```
        LD      A,01001111B   ;RENDI LA PORTA A UN INGRESSO
        OUT     (PIOCRA),A
        LD      A,00001111B   ;RENDI LA PORTA B UN'USCITA
        OUT     (PIOCRB),A
        LD      B,3           ;COUNT = NUMERO DI RIGHE
        LD      C,PIODRB      ;PRENDI IL NUMERO DELLA PORTA DI USCITA
        LD      D,3           ;PRENDI IL NUMERO DELLE COLONNE
        LD      E,11111110B   ;FAI PARTIRE LA CONFIGURAZIONE DI SCANSIONE
                              ;   METTENDO A TERRA LA RIGA ZERO
        LD      H,00000111B   ;PRENDI LA CONFIGURAZIONE DI MASCHERATURA
                              ;   DI TASTIERA
        LD      L,0FFH        ;NUMERO DEL TASTO = -1
FROW:   OUT     (C),E         ;SCANDISCI UNA RIGA
        RLC     E             ;AGGIORNA LA CONFIGURAZIONE DI SCANSIONE
                              ;   PER LA RIGA SUCCESSIVA
        IN      A,(PIODRA)    ;PRENDI IL DATO DI COLONNA DELLA TASTIERA
        AND     H             ;MASCHERA I BIT DI COLONNA
        CP      H             ;C'È QUALCHE COLONNA A TERRA?
        JR      NZ,FCOL       ;SÌ, TROVA QUALE
        LD      A,L           ;NO, AGGIORNA IL NUMERO DI TASTO PER
                              ;   LA RIGA SUCCESSIVA
        ADD     A,D
        LD      L,A
        DJNZ    FROW          ;ESAMINA LA RIGA SUCCESSIVA SE C'È ANCORA
        INC     L             ;IDENTIFICA IL CASO DI NESSUN TASTO TROVATO
        JR      DONE
FCOL:   INC     L             ;INCREMENTA IL NUMERO DEL TASTO
        RRA                   ;QUESTA COLONNA È A TERRA?
        JR      NC,FCOL       ;NO, ESAMINA LA COLONNA SUCCESSIVA
DONE:   HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)		Istruzione (Codice Operativo)	
0000	3E		LD	A,01001111B
0001	4F			
0002	D3		OUT	(PIOCRA),A
0003	PIOCRA			
0004	3E		LD	A,00001111B
0005	0F			
0006	D3		OUT	(PIOCRB),A
0007	PIOCRB			
0008	06		LD	B,3
0009	03			
000A	0E		LD	C,PIODRB
000B	PIODRB			
000C	16		LD	D,3
000D	03			
000E	1E		LD	E,11111110B
000F	FE			
0010	26		LD	H,00000111B
0011	07			
0012	2E		LD	L,0FFH
0013	FF			
0014	ED	FROW:	OUT	(C),E
0015	59			
0016	CB		RLC	E
0017	03			
0018	DB		IN	A,(PIODRA)
0019	PIODRA			
001A	A4		AND	H
001B	BC		CP	H
001C	20		JR	NZ,FCOL
001D	08			
001E	7D		LD	A,L
001F	82		ADD	A,D
0020	6F		LD	L,A
0021	10		DJNZ	FROW
0022	F1			
0023	2C		INC	L
0024	18		JR	DONE
0025	04			
0026	2C	FCOL:	INC	L
0027	1F		RRA	
0028	30		JR	NC,FCOL
0029	FC			
002A	76		HALT	

Ogni volta che la scansione di una riga sbaglia, dobbiamo sommare il numero di colonna al numero del tasto in modo di spostarci al di là della riga presente (provatelo sulla tastiera di Figura 11-22).

Quale sarà il risultato del programma se non si preme nessun tasto? Si noti l'istruzione supplementare INC L tale che il programma si differenzia tra nessun tasto premuto e ultimo tasto premuto. Quale sarà il valore finale dell'Accumulatore per questi due casi? Si noti che il flag Zero potrebbe essere usato anche per distinguere il caso in cui nessun tasto è stato premuto. Potete spiegare come?

Un approccio alternativo potrebbe essere di usare il PIO nel suo modo di controllo in modo che le linee possano essere cambiate da ingressi in uscite. La procedura potrebbe essere:

1) Mettere a terra tutte le colonne e salvare gli ingressi di riga.
2) Mettere a terra tutte le righe e salvare gli ingressi di colonna.
3) Usare insieme gli ingressi di riga e di colonna per determinare il numero del tasto da una tabella.

Provate a scrivere un programma per implementare questa procedura.

Questo programma può essere generalizzato realizzando il numero di righe, il numero di colonna e la combinazione di mascheratura con parametri designati con le pseudo-operazioni EQU.

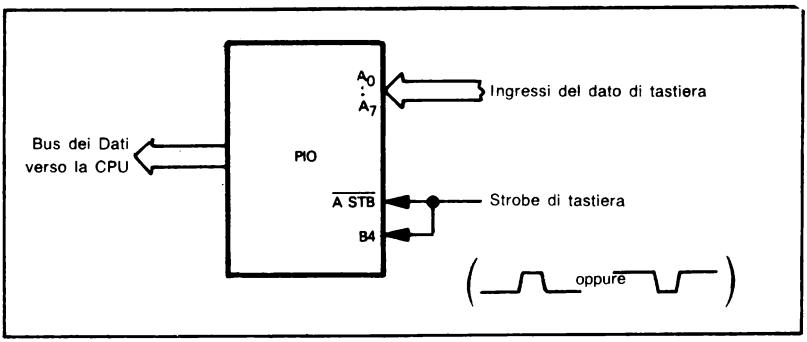


Figura 11-23. Realizzazione dell'Interfaccia I/O per una Tastiera Codificata

Una Tastiera Codificata

Scopo: Prendere dati, quando è possibile, da una tastiera codificata che fornisce un segnale di strobe per ogni trasferimento di dato.

Una tastiera codificata fornisce un unico codice per ogni tasto. Essa possiede una circuiteria elettronica interna che realizza la procedura di scansione e di identificazione dell'esempio precedente. Il compromesso sta tra il software più semplice richiesto dalla tastiera codificata e il minor costo della tastiera non codificata.

Tastiere codificate possono usare matrici di diodi, codificatori TTL o codificatori MOS. I codici possono essere ASCII, EBCDIC o un codice su misura. Memorie PROM fanno spesso parte della circuiteria di codifica.

ROLLOVER

La circuiteria di codifica può fare più che codificare le chiusure dei tasti. Può pure eliminare i rimbalzi dei tasti e gestire il «rollover», il problema relativo al fatto che nello stesso istante possono essere abbassati più tasti. Modi comuni per gestire il rollover sono: «rollover a 2 tasti», per cui due tasti (ma non più di due) premuti nello stesso istante sono visti come chiusure separate e «rollover ad n tasti» per cui qualsiasi numero di tasti contemporaneamente pigiati sono visti come chiusure separate.

La tastiera codificata dà pure un segnale di strobe per ogni trasferimento di dato. Lo strobe indica che è avvenuta una nuova chiusura. La Figura 11-23 mostra l'interfaccia tra una tastiera codificata e il microelaboratore Z80. Il fronte di salita dello strobe incamera il dato nella porta di ingresso. Colleghiamo lo strobe anche alla porta B del PIO in modo che la CPU possa determinare quando si verifica un fronte di salita. Naturalmente la porta B del PIO potrebbe contenere segnali di stato da otto porte. Il software dovrebbe quindi determinare quale porta era attiva con una operazione di scorrimento e di mascheratura.

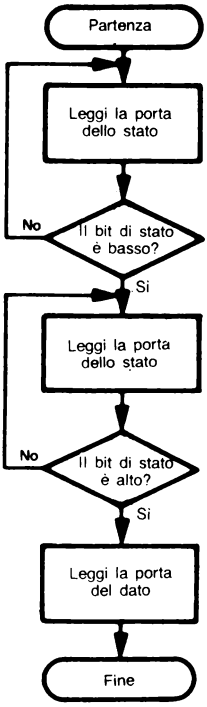
Abbiamo supposto nel programma che il segnale di strobe sia lungo sufficientemente da essere gestito in software dalla CPU. Se non fosse così, il segnale deve essere messo in un registro ed azzerato (con RDY) quando avviene un trasferimento di ingresso o di uscita.

Potreste dover guardare la polarità dello strobe, poichè il PIO reagisce sempre ad un fronte di salita. Potrebbe essere necessaria una porta invertente.

Compito: Ingresso da una tastiera

Scopo: Attendere il gradino di salita di un segnale di strobe sulla Porta B di un PIO e quindi mettere il dato dalla Porta A nell'Accumulatore.

Diagramma di flusso:



L'hardware deve mantenere le linee di controllo in uno stato logico «1» durante il reset per prevenire il posizionamento occidentale dei flag di stato.

Programma Sorgente:

```
        LD      A,01001111B   ;RENDI LA PORTA A UN INGRESSO
        OUT     (PIOCRA),A
        LD      A,11001111B   ;RENDI LA PORTA B UN CONTROLLO
        OUT     (PIOCRB),A
        LD      A,0FFH        ;TUTTE LE LINEE DELLA PORTA B SONO INGRESSI
        OUT     (PIOCRB),A
SRCHL:  IN      A,(PIODRB)    ;ESAMINA LA PORTA DI STATO
        BIT     STB,A         ;LA LINEA DI STROBE È ANDATA BASSA?
        JR      NZ,SRCHL      ;NO, ASPETTA FINCHÈ NON LO SIA
SRCHH:  IN      A,(PIODRB)    ;ESAMINA DI NUOVO LA PORTA DI STATO
        BIT     STB,A         ;È STATO TROVATO UN FRONTE DI SALITA?
        JR      Z,SRCHH       ;NO, ASPETTA FINCHÈ NON SE NE VERIFICHI UNO
        IN      A,(PIODRA)    ;SÌ, VAI A PRENDERE IL DATO
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)		Istruzione (Codice Operativo)	
0000	3E		LD	A,01001111B
0001	4F			
0002	D3		OUT	(PIOCRA),A
0003	PIOCRA			
0004	3E		LD	A,11001111B
0005	CF			
0006	D3		OUT	(PIOCRB),A
0007	PIOCRB			
0008	3E		LD	A,0FFH
0009	FF			
000A	D3		OUT	(PIOCRB),A
000B	PIOCRB			
000C	DB	SRCHL:	IN	A,(PIODRB)
000D	PIODRB			
000E	CB		BIT	STB,A
000F	STB			
0010	20		JR	NZ,SRCHL
0011	FA			
0012	DB	SRCHH:	IN	A,(PIODRB)
0013	PIODRB			
0014	CB		BIT	STB,A
0015	STB			
0016	28		JR	Z,SRCHH
0017	FA			
0018	DB		IN	A,(PIODRA)
0019	PIODRA			
001A	76		HALT	

Se la CPU ripete questa routine, non preleverà nessun altro carattere finché non si verifichi il prossimo fronte di salita sulla linea di strobe. Un livello alto continuo sulla linea di strobe sarà ignorato.

STB dipende da quale bit della Porta B che si usa. La Figura 11-23 mostra che si usa il bit 4, ma i bit 0, 6 e 7 sono, di solito, più facili da esaminare. Provate a riscrivere il programma per usare le posizioni di bit più accessibili.

Il secondo byte delle istruzioni Bit dipende dal valore di STB, ma non è uguale a quel valore. Per esempio, il secondo byte è $4F_{16}$ se STB = 1, 57_{16} se STB = 2, etc.

Convertitore Digitale-Analogico

Scopo: Trasmettere dati ad un convertitore digitale-analogico, che ha un'abilitazione attiva bassa.

I convertitori digitale-analogico producono i segnali continui richiesti da solenoidi, relé, attuatori ed altri dispositivi di uscita elettrici e meccanici. Convertitori tipici sono costituiti da interruttori e da reti di resistori di apportuni valori di resistenze[7]. L'utilizzatore generalmente deve fornire una tensione di riferimento ed un po' di altra circuiteria digitale ed analogica, anche se stanno diventando disponibili unità complete a basso costo.

La Figura 11-24 descrive il convertitore D/A ad 8 bit NE5018 della Signetics, che contiene un registro parallelo ad 8 bit per l'ingresso del dato. Un livello basso sull'ingresso $\overline{LE}$ (Latch Enable) fa entrare il dato di ingresso nei registri, dove rimane dopo che $\overline{LE}$ torna alto.

La Figura 11-25 illustra l'inferfacciamento del dispositivo ad un microelaboratore Z80. Qui il lato A del PIO è usato per generare il segnale Latch Enable. La linea RDY del PIO potrebbe essere usata collegata alla linea $\overline{STB}$ per ottenere un impulso alla fine di ogni ciclo di clock. Tuttavia un solo ciclo di clock può essere non sufficientemente lungo, poichè l'NE5018 richiede un impulso di 400 ns. Inoltre la polarità è l'opposto di quella necessaria per l'NE5018.

Si noti che il PIO incamera il dato d'uscita. Questi perciò rimane stabile durante e dopo la conversione. Il convertitore richiede tipicamente solo alcuni microsecondi per dare un'uscita analogica. In tale modo il registro del convertitore potrebbe essere lasciato disabilitato se la porta non venisse utilizzata per un altro scopo.

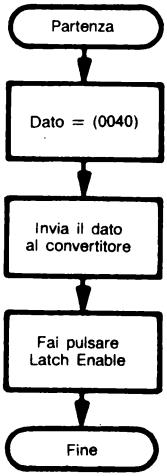
In applicazioni dove non bastano 8 bit di risoluzione, si possono usare convertitori da 10 a 16 bit. Si richiede una logica combinatoria per far passare tutti i bit del dato; alcuni convertitori forniscono parte di questa logica.

Qui il PIO serve sia come porta parallela di dato sia come porta seriale di controllo. Naturalmente se si usa la Porta A per il controllo, si potrebbe gestire in realtà fino ad otto bit.

Compito: Uscita verso il convertitore

Scopo: Inviare dati dalla locazione di memoria 0040 al convertitore.

Diagramma di Flusso:



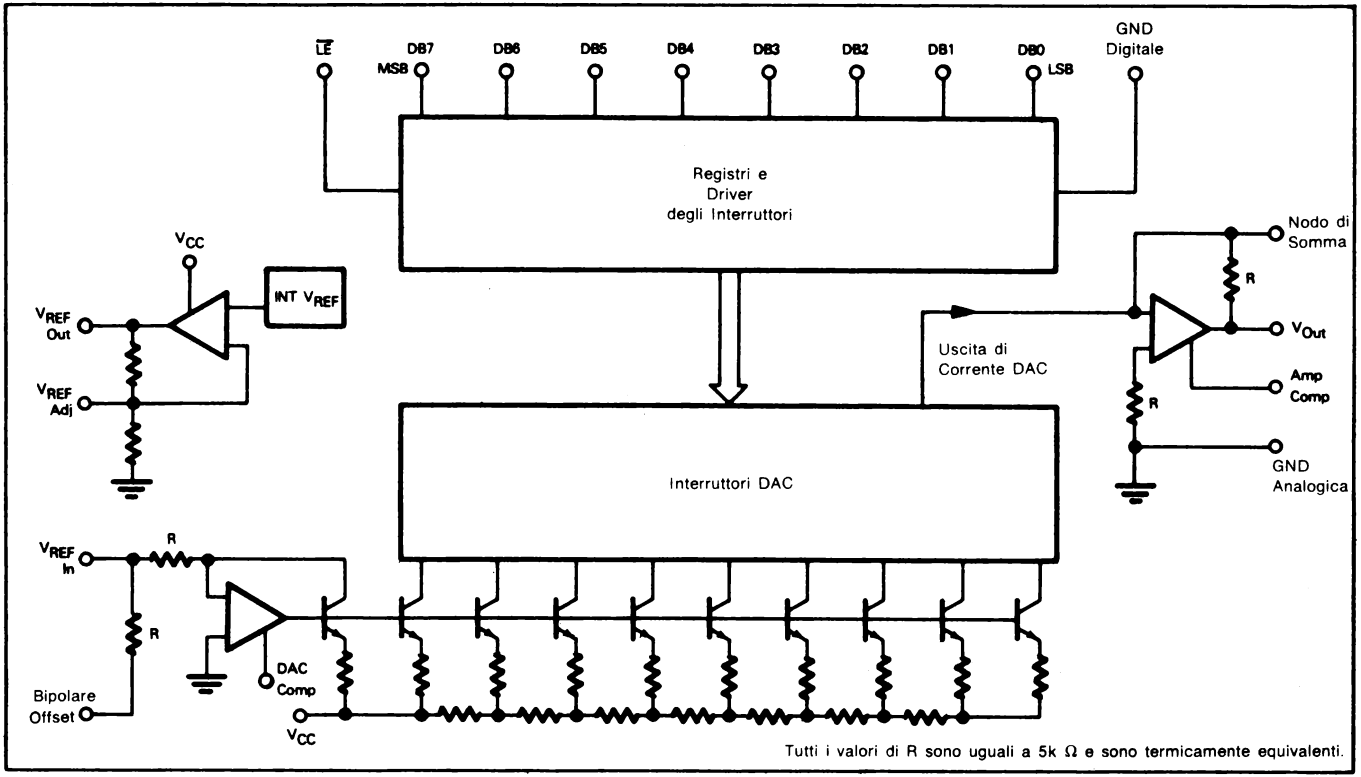


Figura 11-24. Convertitore D/A Signetics NE5018

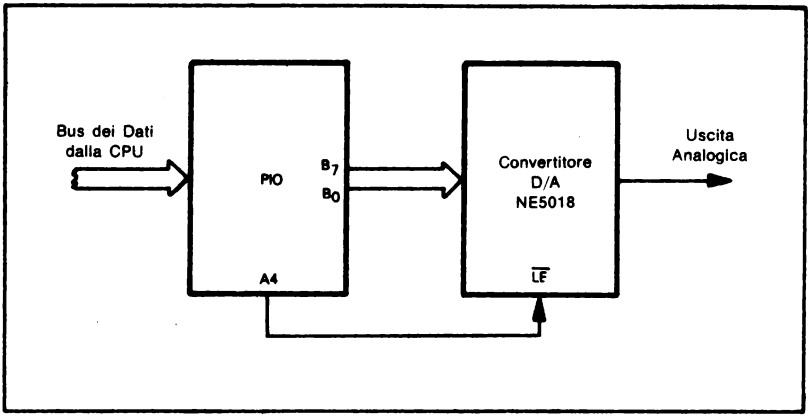


Figura 11-25. Interfaccia per un Convertitore Analogico-Digitale ad 8 Bit

Programma Sorgente:

```
LD      A,11001111B   ;RENDI LA PORTA A UN CONTROLLO
OUT     (PIOCRA),A
SUB     A             ;TUTTI I PIN DELLA PORTA A SONO USCITE
OUT     (PIOCRA),A
LD      A,00001111B   ;RENDI LA PORTA B UN'USCITA
OUT     (PIOCRB),A
LD      A,(40H)       ;PRENDI IL DATO
OUT     (PIODRB),A    ;INVIA IL DATO AL DAC
IN      A,(PIODRA)    ;PRENDI IL VECCHIO DATO DI CONTROLLO
RES     4,A           ;TIENI BASSO LATCH ENABLE
OUT     (PIODRA)A
SET     4,A           ;TIENI ALTO LATCH ENABLE
OUT     (PIODRA),A
HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Operativo)	
0000	3E	LD	A,11001111B
0001	CF		
0002	D3	OUT	(PIOCRA),A
0003	PIOCRA		
0004	97	SUB	A
0005	D3	OUT	(PIOCRA),A
0006	PIOCRA		
0007	3E	LD	A,00001111B
0008	0F		
0009	D3	OUT	(PIOCRB),A
000A	PIOCRB		
000B	3A	LD	A,(40H)
000C	40		
000D	00		
000E	D3	OUT	(PIODRB),A
000F	PIODRB		
0010	DB	IN	A,(PIODRA)
0011	PIODRA		
0012	CB	RES	4,A
0013	A7		
0014	D3	OUT	(PIODRA),A
0015	PIODRA		
0016	CB	SET	4,A
0017	E7		
0018	D3	OUT	(PIODRA),A
0019	PIODRA		
001A	76	HALT	

Il bit particolare da posizionare ad 1 ed a 0 dipende, naturalmente, da come si collega il Latch Enable alla porta di controllo. Spesso è conveniente usare il bit 0 per scopi di controllo poichè, se quel bit è stato originariamente azzerato, può essere posizionato ad 1 con un'istruzione INC e azzerato con un'istruzione DEC.

Potremmo usare il breve segnale di strobe automatico proveniente da BACK se il segnale Latch Enable era attivo alto (e se questo strobe era sufficientemente lungo con BACK collegato a $\overline{\text{B STB}}$). Il programma allora diventerebbe:

```
LD      A,00001111B   ;RENDI LA PORTA B UN'USCITA
OUT     (PIOCRB),A
LD      A,(40H)       ;PRENDI IL DATO
OUT     (PIODRB),A    ;INVIA IL DATO AL DAC ED ABILITA IL REGISTRO
HALT
```

Una porta invertente potrebbe generare un segnale attivo basso. Si noti come siano necessarie così poche istruzioni.

Convertitore Analogico-Digitale

Scopo: Prendere dati da un convertitore analogico-digitale ad 8 bit che richiede un impulso Initiate Conversion per dare inizio al processo di conversione e che ha una linea Data Valid per indicare il completamento del processo e la disponibilità del dato valido.

I convertitori analogico-digitale gestiscono i segnali continui prodotti da vari tipi di sensori e di trasduttori[8]. Il convertitore produce l'ingresso digitale che il computer richiede.

Un tipo di convertitore analogico-digitale è il dispositivo ad approssimazione successiva, che fa un confronto diretto di un bit durante ogni ciclo di clock. Tali convertitori sono veloci ma hanno una piccola immunità al rumore. Un altro tipo di convertitore analogico-digitale è il convertitore ad integrazione a doppia pendenza. Questi dispositivi impiegano più tempo ma sono più insensibili al rumore. Si usano pure altre tecniche, quale la tecnica a bilanciamento di carica incrementale.

Usualmente i convertitori analogico-digitale richiedono della circuiteria digitale ed analogica esterna, anche se stanno diventando disponibili unità complete a basso costo. La Figura 11-26 mostra il convertitore A/D ad 8 bit 8703 della Teledyne Semiconductor. Il dispositivo contiene un registro del risultato ed uscite del dato tri-state. Un impulso sulla linea Initiate Conversion dà inizio alla conversione dell'ingresso analogico; dopo circa due millisecondi il risultato andrà ai registri d'uscita e l'uscita Data Valid lo indicherà con un impulso di basso. Il dato è letto dai registri applicando uno «0» all'ingresso $\overline{\text{ENABLE}}$.

La Figura 11-27 mostra l'interfaccia per l'elaborazione Z80 e per il convertitore 8703[9]. Si usa la Porta B per fornire un impulso Initiate Conversione (attivo basso) di lunghezza sufficiente. Il segnale Data Valid è collegato ad A $\overline{\text{STB}}$ così un impulso di basso su di esso farà incamerare il dato convertito nella Porta A. Il segnale Data Valid è collegato anche ad un bit della Porta B così che la CPU possa determinare il suo valore. Il fronte importante sulla linea Data Valid è il fronte da basso ad alto, che indica il completamento della conversione. Come nel caso della tastiera codificata, occorreranno circuiti elettrici aggiuntivi se l'impulso su Data Valid è troppo breve da essere gestito in software. Si noti che qui stiamo usando la Porta B sia per lo stato che per il controllo.

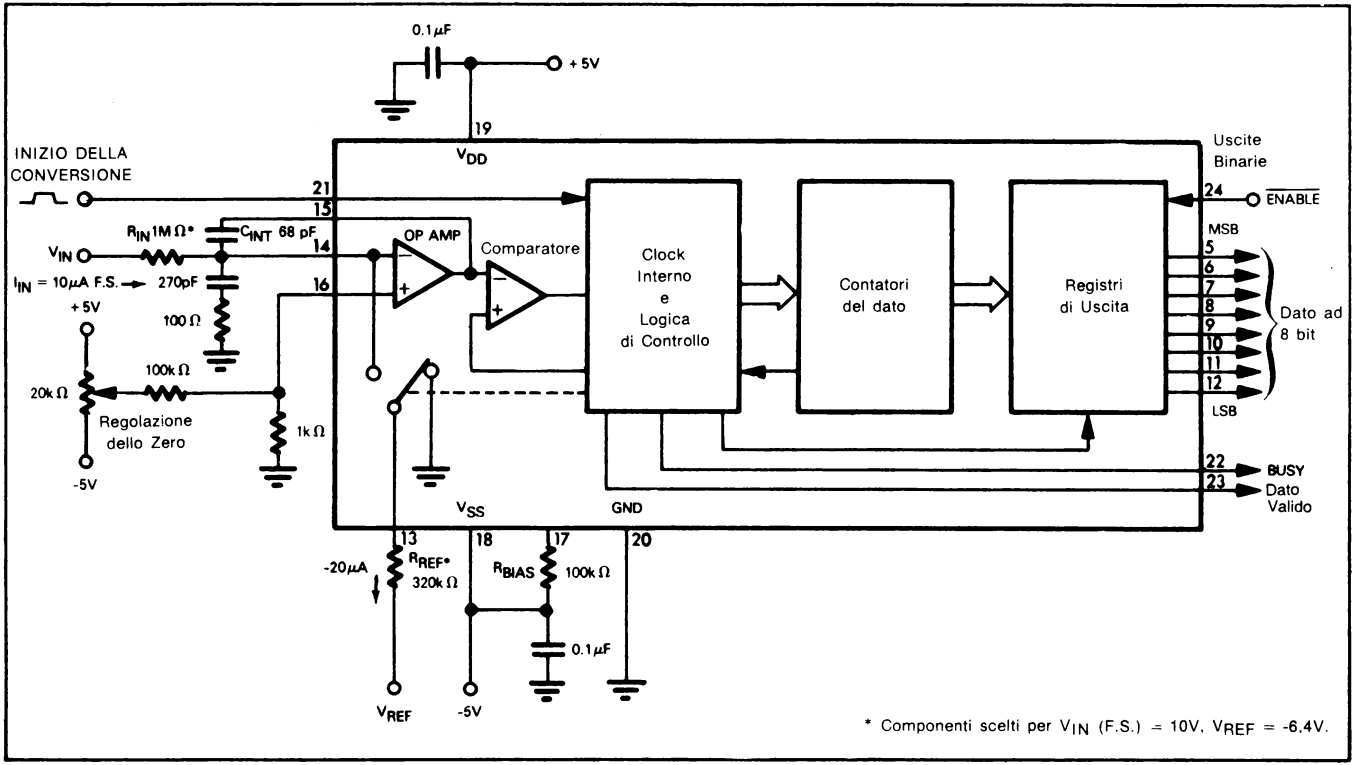


Figura 11-26. Convertitore A/D Teledyne 8703

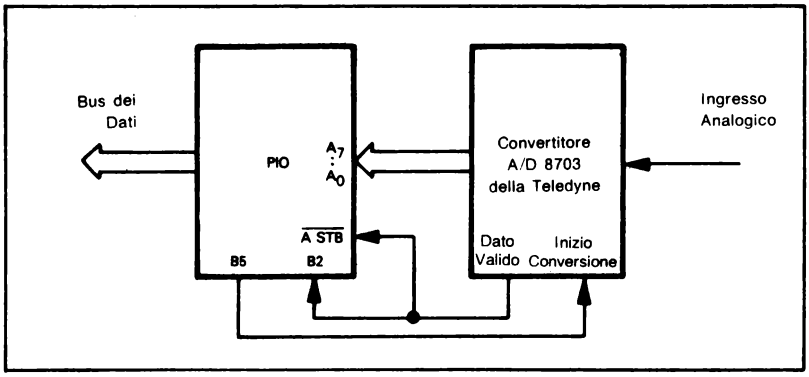
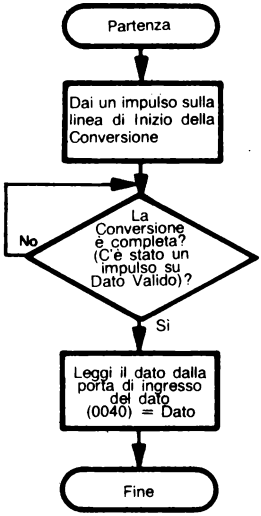


Figura 11-27. Interfaccia per un Convertitore Analogico-Digitale ad 8 Bit

Compito: Ingresso dal convertitore

Scopo: Dare inizio al processo di conversione, attendere che Data Valid vada basso e poi alto, e quindi leggere il dato e memorizzarlo nella locazione di memoria 0040.

Diagramma di Flusso:



Si noti che qui il PIO serve da porta di dato parallelo, da porta di stato e di controllo seriale.

Programma Sorgente:

```
        LD      A,01001111B   ;RENDI LA PORTA A UN INGRESSO
        OUT     (PIOCRA),A
        LD      A,11001111B   ;RENDI LA PORTA B UN CONTROLLO
        OUT     (PIOCRB),A
        LD      A,00001111B   ;B4-7 USCITE, B0-3 INGRESSI
        OUT     (PIOCRB),A
        LD      A,00100000B   ;INVIA UN ALTO SU INIZIO DELLA CONVERSIONE
        OUT     (PIODRB)A
        SUB     A             ;INVIA UN BASSO SU INIZIO DELLA CONVERSIONE
        OUT     (PIODRB),A
WTLOW:  IN      A,(PIODRB)    ;LA LINEA DATO VALIDO È ANDATA BASSA?
        BIT     2,A
        JR      NZ,WTLOW      ;NO, ATTENDI
WTHI:   IN      A,(PIODRB)    ;IL DATO È DISPONIBILE?
        BIT     2,A
        JR      Z,WTLOW       ;NO, ATTENDI
        IN      A,(PIODRA)    ;SÌ, VAI A PRENDERE IL DATO DAL CONVERTITORE
        LD      (40H),A       ;SALVA IL DATO DEL CONVERTITORE
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Operativo)	
0000	3E	LD	A,01001111B
0001	4F		
0002	D3	OUT	(PIOCRA),A
0003	PIOCRA		
0004	3E	LD	A,11001111B
0005	CF		
0006	D3	OUT	(PIOCRB),A
0007	PIOCRB		
0008	3E	LD	A,00001111B
0009	0F		
000A	D3	OUT	(PIOCRB),A
000B	PIOCRB		
000C	3E	LD	A,00100000B
000D	20		
000E	D3	OUT	(PIODRB),A
000F	PIODRB		
0010	97	SUB	A
0011	D3	OUT	(PIODRB),A
0012	PIODRB		
0013	DB	WTLOW: IN	A,(PIODRB)
0014	PIODRB		
0015	CB	BIT	2,A
0016	57		
0017	20	JR	NZ,WTLOW
0018	FA		
0019	DB	WTHI: IN	A,(PIODRB)
001A	PIODRB		
001B	CB	BIT	2,A
001C	57		
001D	28	JR	Z,WTHI
001E	FA		
001F	DB	IN	A,(PIODRA)
0020	PIODRA		
0021	32	LD	(40H),A
0022	40		
0023	00		
0024	76	HALT	

Un approccio per configurare i PIO è l'uso della istruzione di Uscita di Blocco ripetuta OTIR e di una tabella di memoria contenente le parole da inviare al registro di Controllo. Una routine tipica potrebbe essere:

```
LD      B,LENG      ;COUNT = NUMERO DELLE PAROLE DI CONTROLLO
LD      C,PIOCR     ;PRENDI IL NUMERO DELLA PORTA DI CONTROLLO
LD      HL,CTLTAB   ;INIZIO DEGLI INDIRIZZI DELLA TABELLA
                    ;   DI CONTROLLO DEL PIO
OTIR                ;CONFIGURA IL PIO
```

Infatti, si potrebbe usare un'altra tabella (o lo Stack) per mantenere il numero delle parole di controllo ed il numero di porta di ogni PIO.

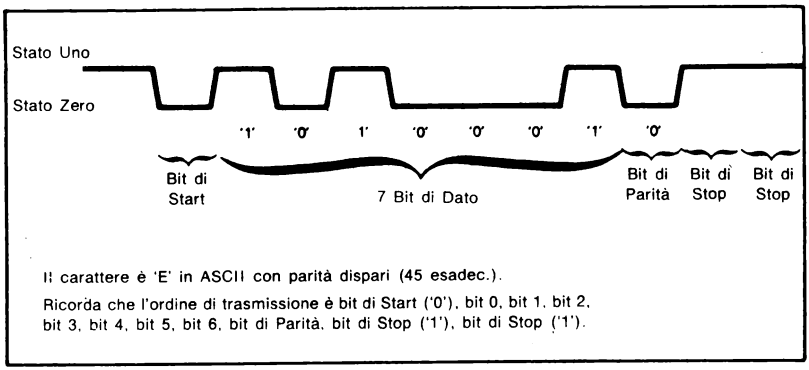


Figura 11-28. Formato Dati di Telescrivente

Telescrivente (TTY)

INTERFACCIA CON TTY

Scopo: Trasferire dati a e da una telescrivente standard seriale a 10 caratteri al secondo.

La telescrivente standard trasferisce dati in un modo seriale asincrono. La procedura è la seguente:

TTY STANDARD

FORMATO DEL CARATTERE

1) La linea normalmente è nello stato uno.
2) Un bit di Start (bit zero) precede ogni carattere.
3) Il carattere è di solito un codice ASCII a 7 bit col bit meno significativo trasmesso per primo.
4) Il bit più significativo è un bit di Parità, che può essere pari, dispari o fisso a zero o a uno.
5) Due bit di stop (uno logico) seguono ogni carattere.

La Figura 11-28 mostra il formato. Si noti che ogni carattere richiede la trasmissione di undici bit, dei quali solo sette contengono informazioni. Poiché la velocità del dato è dieci caratteri al secondo la velocità dei bit è 10X11 o 110 Baud. Ogni bit perciò ha una durata di 1/110 di secondo o 9,1 millisecondi. Questa durata è una media: la telescrivente non la mantiene ad un alto livello di precisione.

Affinché una telescrivente comunichi in maniera adeguata con un computer sono necessarie le seguenti procedure:

Ricezione (diagramma di flusso di Figura 11-29):

MODO DI RICEZIONE DELLA TTY

Passo 1) Cercare un bit di Start (uno zero logico) sulla linea del dato.
Passo 2) Centrare la ricezione aspettando metà tempo, o 4,55 millisecondi.
Passo 3) Prendere i bit del dato, attendere la durata di un bit prima di prendere ogni bit. Riunire i bit del dato in una parola facendo scorrere dapprima il bit verso il Carry e quindi facendo scorrere circolarmente il dato col Carry. Ricordatevi che per primo si riceve il bit meno significativo.
Passo 4) Generare la Parità in ricezione e controllarla rispetto alla Parità trasmessa. Se non coincidono, indicare un «errore di Parità».
Passo 5) Prendere i bit di Stop (attendere la durata di un bit tra due ingressi). Se non sono corretti (se entrambi i bit di Stop non sono ad uno), indicare un «errore di framing».

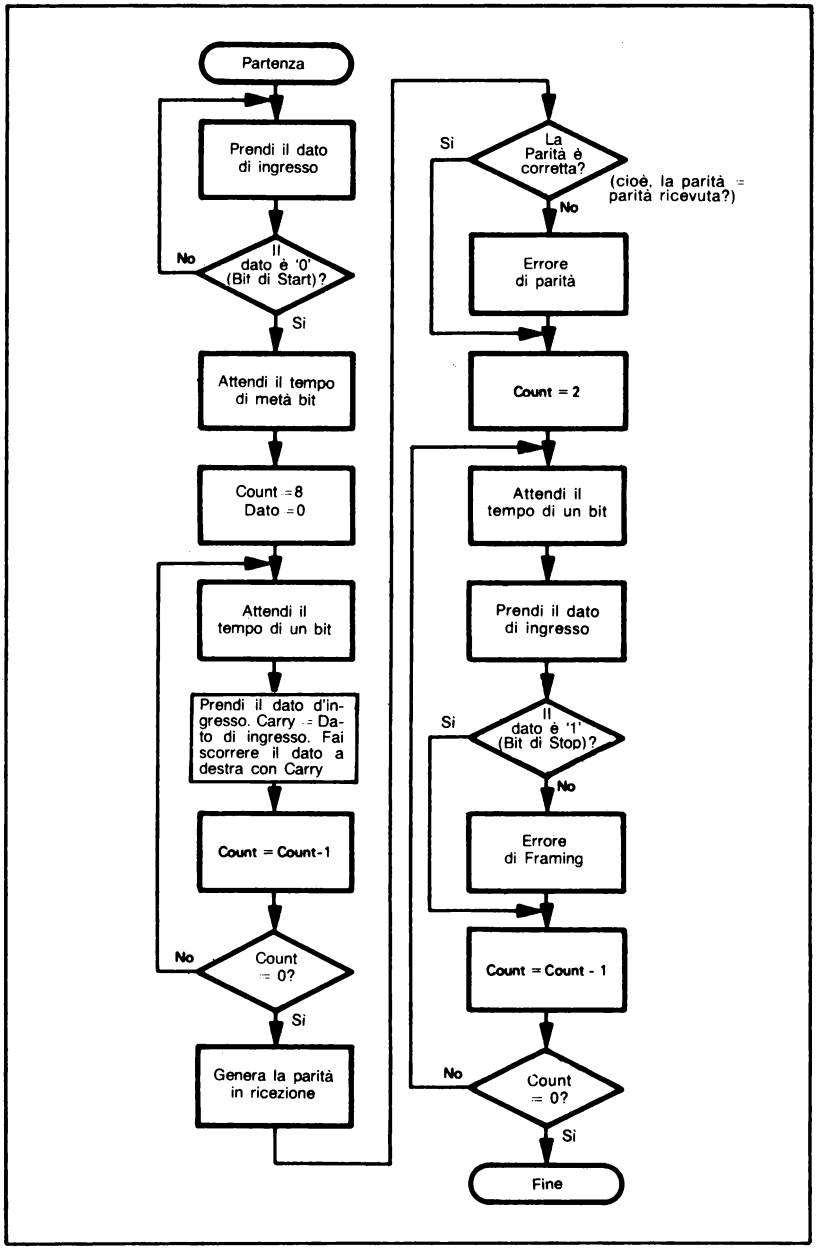


Figura 11-29. Diagramma di Flusso per la Procedura di Ricezione

Compito 1: Leggere un dato

Scopo: Prendere un dato da una telescrivente mediante il bit 7 di una porta di un PIO e metterlo nella locazione di memoria 0060. Per la procedura, vedere la Figura 11-29.

Programma Sorgente:

(Supponiamo che la porta seriale sia il bit 7 del PIO e che non siano necessari né parità né controllo di framing).

```
        LD      A,01001111B  ;RENDI LA PORTA A UN INGRESSO
        OUT     (PIOCRA),A
WTSTB:  IN      A,(PIODRA)   ;LEGGI LA LINEA SERIALE
        RLA                  ;C'È UN BIT DI START?
        JR      C,WTSTB      ;NO, ASPETTA CHE CI SIA
        CALL    DHALF        ;SÌ, RITARDA IL TEMPO
                             ;   DI METÁ BIT PER CENTRARLO
        LD      D,10000000B  ;BIT IN MSB DI COUNT
RCVB:   CALL    DFULL        ;ATTENDI IL TEMPO DI UN BIT
        IN      A,(PIODRA)   ;LEGGI LA LINEA SERIALE
        RLA                  ;SPOSTA IL BIT IN CARRY
        RR      D            ;SPOSTA IL BIT NELLA PAROLA ASSEMBLATA
        JR      NC,RCVB      ;CONTINUA SE IL BIT DI COUNT
                             ;   NON È IN CARRY
        LD      A,D
        LD      (60H),A
        HALT
```

(Programma di ritardo)

```
        ORG     30H
DHALF:  PUSH    DE           ;SALVA I VECCHI REGISTRI
        LD      D,8          ;CONTA LA LUNGHEZZA DI MEZZO BIT
        JR      DLY16
DFULL:  PUSH    DE           ;SALVA I VECCHI REGISTRI
        LD      D,16         ;CONTA LA LUNGHEZZA DI UN BIT
DLY16:  LD      E,8DH        ;RITARDA IL TEMPO DI 1/16 DI BIT
DLY1:   DEC     E
        JR      NZ,DLY1
        DEC     D
        JR      NZ,DLY16
        POP     DE           ;RIPRISTINA I VECCHI REGISTRI
        RET
```

Ricordatevi che si riceve per primo il bit 0 del dato.

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Operativo)		
0000	3E		LD	A,01001111B
0001	4F			
0002	D3		OUT	(PIOCRA),A
0003	PIOCRA			
0004	DB	WTSTB:	IN	A,(PIODRA)
0005	PIODRA			
0006	17		RLA	
0007	38		JR	C,WTSTB
0008	FB			
0009	CD		CALL	DHALF
000A	30			
000B	00			
000C	16		LD	D,10000000B
000D	80			
000E	CD	RCVB:	CALL	DFULL
000F	35			
0010	00			
0011	DB		IN	A,(PIODRA)
0012	PIODRA			
0013	17		RLA	
0014	CB		RR	D
0015	1A			
0016	30		JR	NC,RCVB
0017	F6			
0018	7A		LD	A,D
0019	32		LD	(60H),A
001A	60			
001B	00			
001C	76		HALT	
0030	D5	DHALF:	PUSH	DE
0031	16		LD	D,8
0032	08			
0033	18		JR	DLY16
0034	03			
0035	D5	DFULL:	PUSH	DE
0036	16		LD	D,16
0037	10			
0038	1E	DLY16:	LD	E,8DH
0039	8D			
003A	1D	DLY1:	DEC	E
003B	20		JR	NZ,DLY1
003C	FD			
003D	15		DEC	D
003E	20		JR	NZ,DLY16
003F	F8			
0040	D1		POP	DE
0041	C9		RET	

Questo programma presuppone che si possa usare lo Stack per le chiamate di subroutine, per cui il programma monitor deve inizializzare il Puntatore dello Stack. Altrimenti dovete inizializzare il Puntatore dello Stack come mostrato nel Capitolo 10.

Le costanti per la routine di ritardo sono state calcolate proprio come indicato prima in questo capitolo. Potete provare a determinarle voi stessi. I ritardi non devono essere molto precisi perché la ricezione è fatta nel centro, i messaggi sono brevi, la velocità dei bit è bassa e la telescrivente stessa non è molto precisa.

Come potreste ampliare questo programma per controllare i due bit di stop? Essi devono essere entrambi ad uno altrimenti si è verificato un errore di framing.

Potete ampliare questo programma per controllare la parità dispari sostituendo l'istruzione LD A,D con la sequenza:

```
SUB     A
AND     D           ;LA PARITÀ È DISPARI?
JP      PE,PRERR    ;NO, È AVVENUTO UN ERRORE DI PARITÀ
```

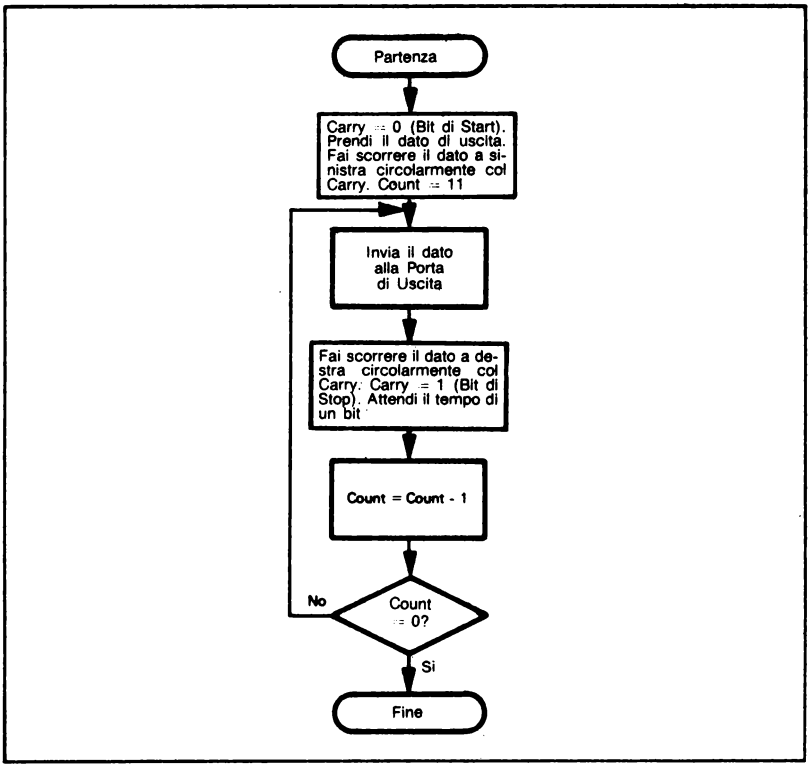


Figura 11-30. Diagramma di Flusso per la Procedura di Trasmissione

Compito 2: Scrivere un dato

Scopo: Trasmettere un dato ad una telescrivente mediante il bit 0 di un registro di dato di PIO. Il dato si trova nella locazione di memoria 0060.

Trasmissione (diagramma di flusso di Figura 11-30)

MODO DI TRASMISSIONE DELLA TTY

Passo 1) Trasmettere un bit di Start (cioè un uno logico).
Passo 2) Trasmettere i sette bit del dato, a partire dal bit meno significativo.
Passo 3) Generare e trasmettere il bit di Parità.
Passo 4) Trasmettere due bit di Stop (cioè uni logici).

La routine di trasmissione deve attendere la durata di un bit tra ogni operazione.

Programma Sorgente: (Supponiamo che non si debba generare la parità)

```
;
;RENDI IL PIO UNA PORTA D'USCITA
;
        LD      A,00001111B  ;RENDI LA PORTA B UN'USCITA
        OUT     (PIOCRB),A
;
;PRENDI IL DATO ED AZZERA IL BIT DI START
;
        LD      A,(60H)      ;PRENDI IL DATO
        ADD     A,A          ;FAI UNO SCORRIMENTO A SINISTRA
                             ;  E FORMA IL BIT DI START
        LD      B,11         ;COUNT = 11 BIT
;
;TRASMETTI UN BIT ED AGGIORNA IL DATO
;
TBIT:   OUT     (PIODRB),A   ;TRASMETTI UN BIT
        RRÀ                  ;AGGIORNA COL PROSSIMO BIT
        SCF                  ;FORMA IL BIT DI STOP (UNO LOGICO)
;
;RITARDA 9,1 MS E CONTA I BIT
;
        CALL    DFULL        ;RITARDA 9,1 MS
        DJNZ    TBIT         ;CONTEGGIA GLI 11 BIT
        HALT
```

La subroutine DFULL è la stessa di prima. Ricordatevi che il bit 0 del dato deve essere trasferito per primo.

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)		Istruzione (Codice Operativo)	
0000	3E		LD	A,00001111B
0001	0F			
0002	D3		OUT	(PIOCRB),A
0003	PIOCRB			
0004	3A		LD	A,(60H)
0005	60			
0006	00			
0007	87		ADD	A,A
0008	06		LD	B,11
0009	0B			
000A	D3	TBIT:	OUT	(PIODRB),A
000B	PIODRB			
000C	1F		RRA	
000D	37		SCF	
000E	CD		CALL	DFULL
000F	35			
0010	00			
0011	10		DJNZ	TBIT
0012	F7			
0013	76		HALT	

ADD A,A azzera il bit meno significativo cosicché questi può essere usato come bit di start. Il bit più significativo è salvato nel Carry. In applicazioni reali, la routine di partenza dovrebbe mettere un «1» logico sulla linea della telescrivente dopo la configurazione poichè quella linea potrebbe essere normalmente nello stato di mark (uno).

Ogni carattere consiste di 11 bit, cominciando con un bit di start (zero) e finendo con due bit di stop (uni).

Questo programma può essere facilmente esteso per generare caratteri a 7 bit con parità dispari nel bit più significativo. La routine di generazione della parità (da inserire dopo LD A, (60H)) è:

```
        ANA     A           ;LA PARITÀ È DISPARI
        JP      PO,STBIT    ;SÌ, NESSUN PROBLEMA
        SET     7,A         ;NO, RENDILA DISPARI PONENDO AD 1 L'MSB
STBIT:  ADD     A,A         ;FA SCORRERE A SINISTRA E GENERA IL BIT
                            ;  DI START
```

UART

Come potreste generare la parità pari? Queste procedure sono sufficientemente comuni e complesse da meritare uno speciale dispositivo LSI: l'UART o Universal Asynchronous Receiver/Transmitter[10]. L'UART realizzerà la procedura di ricezione e fornirà il dato in formato parallelo ed un segnale Data Ready. Accetterà pure dati in formato parallelo, realizzerà la procedura di trasmissione e fornirà un segnale Peripheral Ready quando può gestire altri dati. Gli UART possono avere altre caratteristiche, comprendenti:

1) Capacità di gestire varie lunghezze di bit (abitualmente 5 o 8), opzioni di parità e numeri di bit di Stop (di salita 1, 1-1/2 e 2).
2) Indicatori per errori di framing, errori di parità ed «errori di overrun» (errore di lettura di un carattere prima che ne sia stato ricevuto un altro).
3) Compatibilità con l'RS-232: cioè un segnale di uscita Request to Send (RTS) che indica la presenza di dati verso l'equipaggiamento di comunicazione ed un segnale di ingresso Clear to Send (CTS) che indica, in risposta a RTS, lo stato di pronto dell'equipaggiamento di comunicazione. Ci può essere una fornitura di altri segnali RS-232, quali Received Signal Quality, Data Set Ready o Data Terminal Ready.
4) Uscite tri-state e compatibilità di controllo con un microelaboratore.
5) Opzioni di clock che permettono all'UART di campionare i dati d'ingresso parecchie volte per rilevare falsi bit di Start oppure altri errori.
6) Caratteristiche di interrupt e controlli.

L'UART agisce come quattro porte parallele: una porta di dato d'ingresso, una porta di dato d'uscita, una porta di stato d'ingresso ed una porta di controllo d'uscita. I bit di stato comprendono indicatori di errore come pure flag di Ready. I bit di controllo scelgono varie opzioni. Gli UART non sono costose (da $5 a 50$, in funzione delle caratteristiche) e sono facili da usare.

IL DISPOSITIVO DI INGRESSO/USCITA SERIALE DELLO Z80 (SIO)

Il Dispositivo di Ingresso/uscita Seriale dello Z80 (vedere la Figura 11-31) è un controllore di comunicazione completo, progettato specificatamente per usi in microcomputer basati sullo Z80. Può servire per una grande varietà di funzioni di comunicazione, ma tratteremo solo il suo uso come semplice ricevitore/trasmettitore asincrono[12].

FULL-DUPLEX

Il SIO ha due canali completi (A e B) che possono sia ricevere che trasmettere dati seriali (vedere la Figura 11-32). I canali che possono ricevere e trasmettere contemporaneamente sono chiamati full-duplex. Alternative sono il half-duplex (capace di trasmettere e di ricevere, ma non contemporaneamente), il solo ricevitore e il solo trasmettitore.

INDIRIZZI DEL SIO

Un SIO occupa quattro indirizzi di porte d'ingresso e quattro indirizzi di porte d'uscita. Le linee $B/\overline{A}$ (Selezione del Canale A o B) e $C/\overline{D}$ (Selezione del Controllo o del Dato) scelgono una delle quattro porte come descritto nella Tabella 11-7. Molto spesso, i progettisti collegano il bit d'indirizzo A_0 all'ingresso $B/\overline{A}$ e il bit d'indirizzo A_1 all'ingresso $C/\overline{D}$. Perciò il SIO occupa quattro indirizzi di porta consecutivi come descritto nell'ultima colonna della Tabella 11-7.

INDIRIZZAMENTO DEI REGISTRI DI LETTURA E DI SCRITTURA DEL SIO

Come con il PIO, i SIO hanno più registri di controllo che indirizzi. Infatti ogni SIO ha otto registri in ogni canale per il controllo e tre registri per lo stato. La Figura 11-33 contiene diagrammi di ogni registro di controllo o di Write; la Figura 11-34 contiene diagrammi di ogni registro di stato o di Read. Si richiedono due trasferimenti per leggere da o scrivere in ogni registro eccetto che per il Registro di Scrittura 0. Il primo trasferimento (scritto nel Registro di Scrittura 0) contiene tre bit che dirigono il trasferimento successivo al o dal registro selezionato. Si noti, in Figura 11-33, che questi tre bit occupano le posizioni dei tre bit meno significativi e che degli zero nelle altre posizioni dei bit indicano un byte che non ha altra funzione se non quella di indirizzamento.

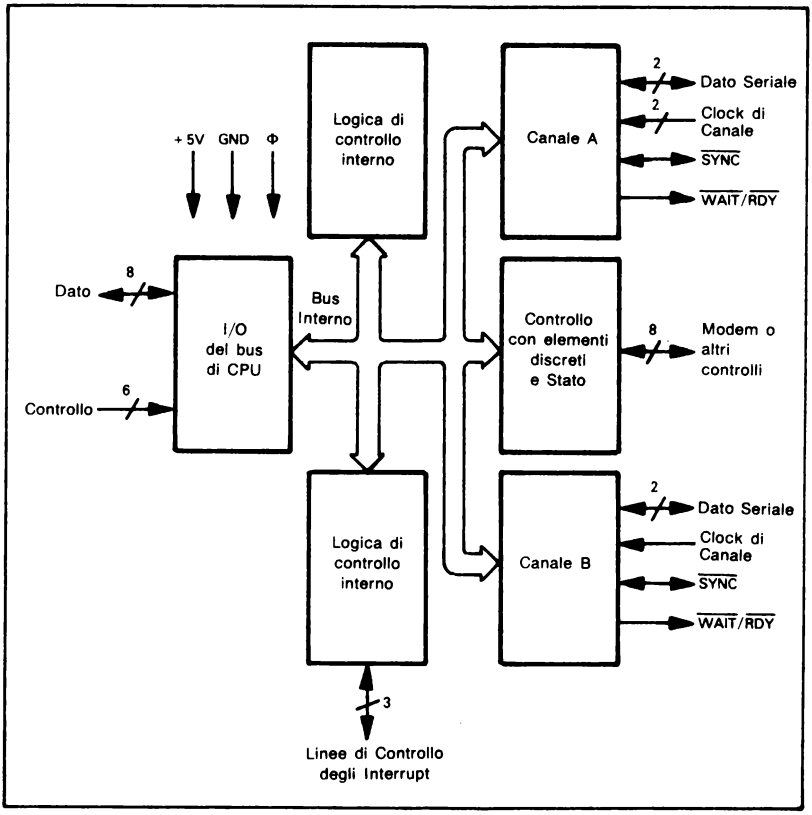


Figura 11-31. Schema a Blocchi dello Z80 SIO

Tabella 11-7. Indirizzi del SIO

CONTROLLO O SELEZIONE DEL DATO	SELEZIONE DEL CANALE B OD A	REGISTRO INDIRIZZATO	INDIRIZZO DI PORTA (A PARTIRE DA SIOADD)
0	0	Registro di Dato A	SIOADD
0	1	Registro di Dato B	SIOADD+1
1	0	Controllo A	SIOADD+2
1	1	Controllo B	SIOADD+3
Gli indirizzi delle porte presuppongono che C/D sia collegato ad A_1 e B/A ad A_0.			

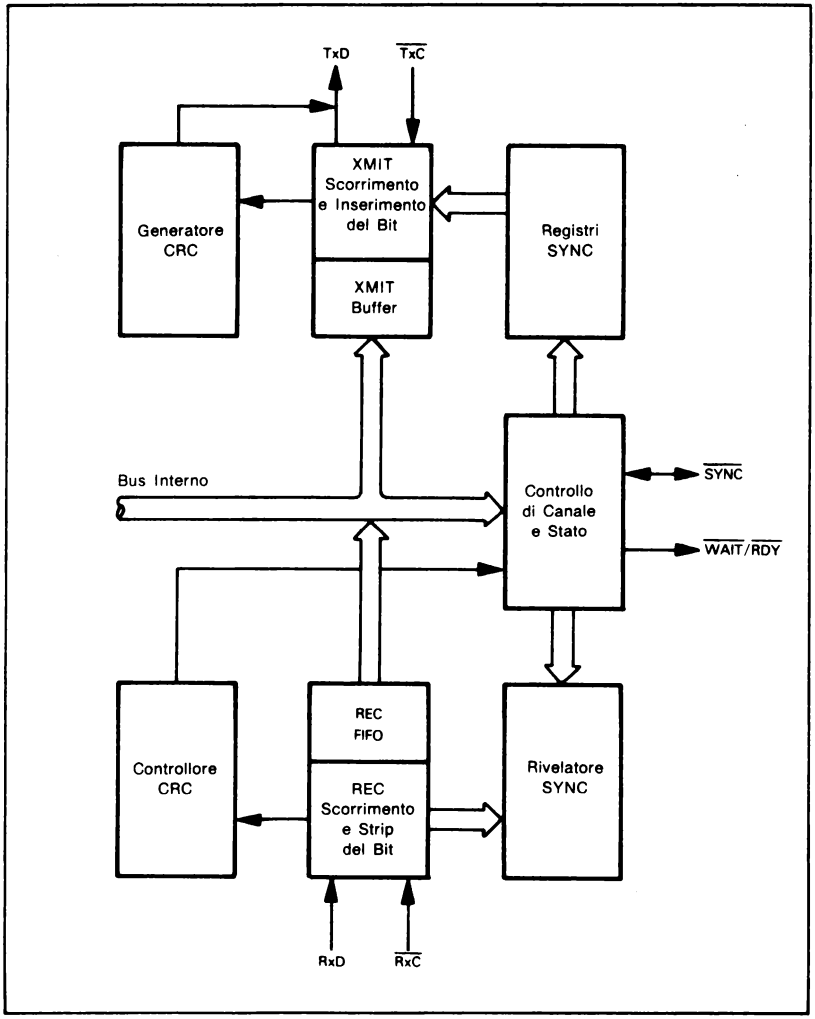


Figura 11-32. Schema a Blocchi del Canale del SIO

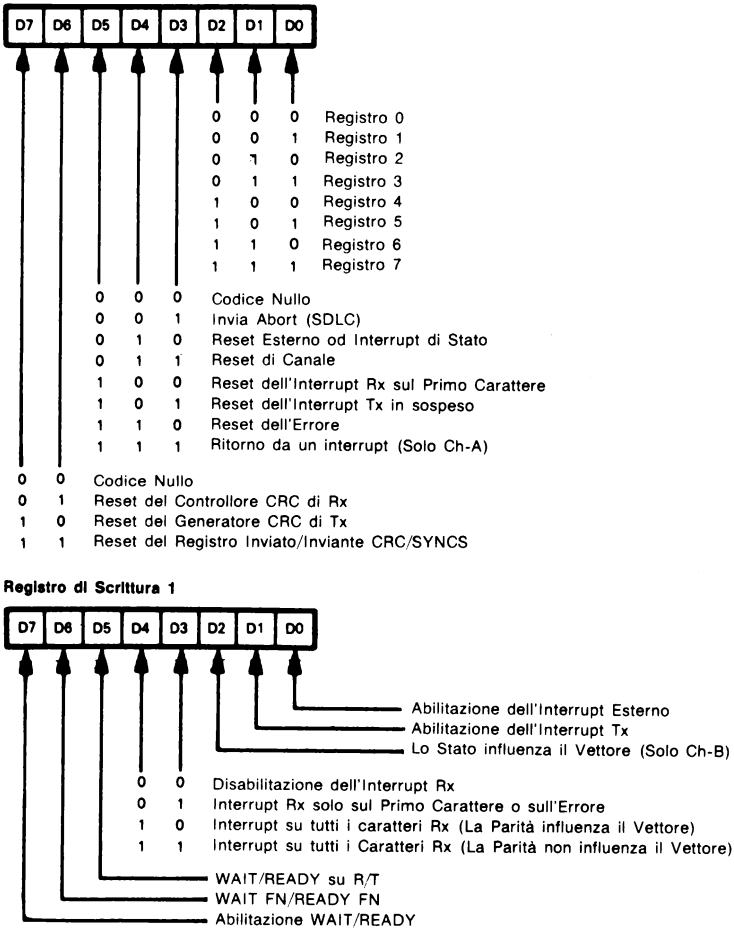
Registri di Scrittura

Il SIO Z80 contiene otto registri in ogni canale che sono programmati (con scrittura) dal software del sistema per configurare la funzione di ogni canale. Tutti i registri di Scrittura, tranne il Registro di Scrittura 0, richiedono due byte che devono essere programmati in maniera opportuna. Il primo byte contiene tre bit che puntano verso il registro selezionato (D0-D2): il secondo byte è la reale parola di controllo che deve essere scritta in quel registro per configurare il SIO.

Il Registro di Scrittura 0 è un caso speciale. RESET (o comando interno o ingresso esterno) inizializzerà il SIO per il Registro di Scrittura 0. Si può accedere ai comandi fondamentali (CMD2-CMD0) e ai controlli CRC con un solo byte usando il Registro di Scrittura 0.

Nel primo byte di qualunque accesso a registro di Scrittura sono contenuti i comandi fondamentali (CMD2-CMD0) e i controlli CRC (CRC0, CRC1), in modo da mantenere il massimo controllo di sistema e la massima flessibilità.

Registro di Scrittura 0



Registro di Scrittura 1

Figura 11-33. Controllo del SIO o Registri di Scrittura

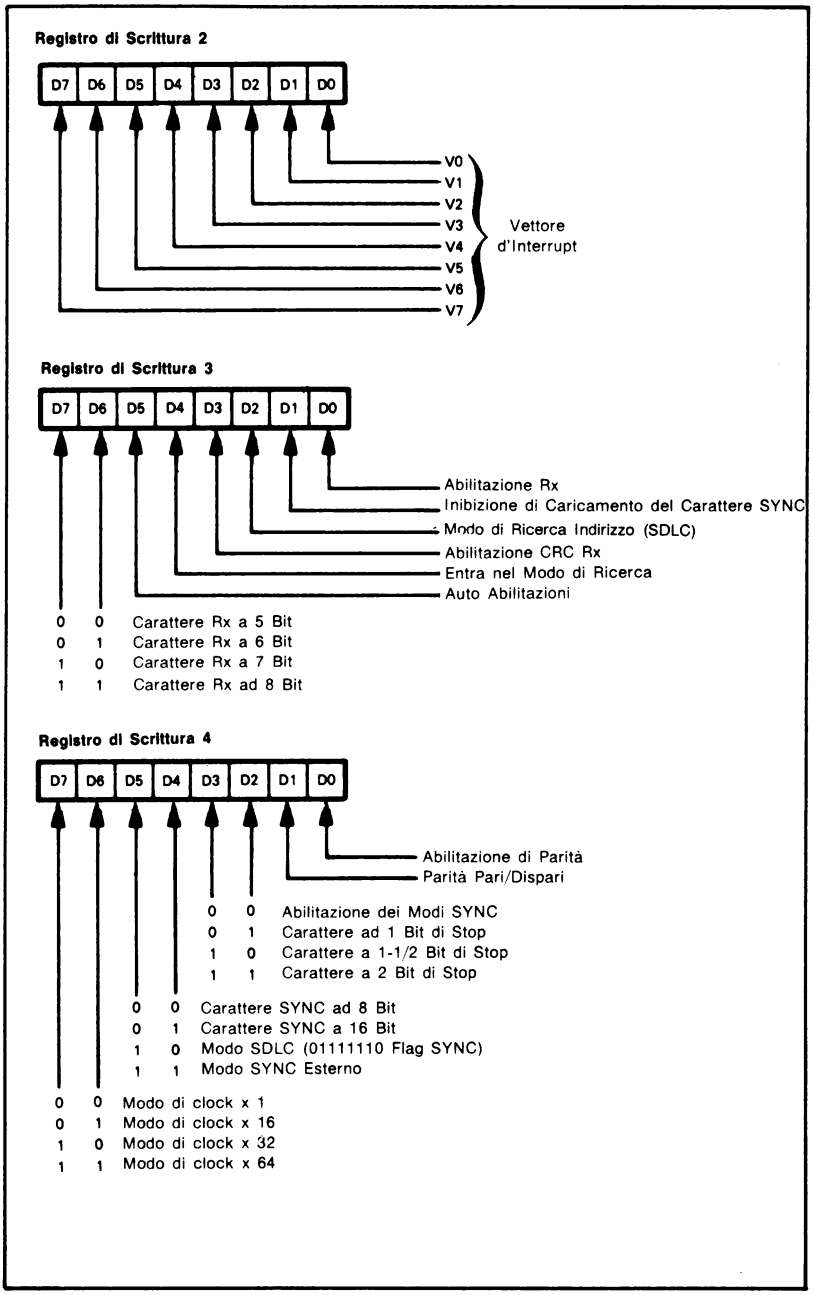


Figura 11-33. Controllo del SIO o Registri di Scrittura (continua)

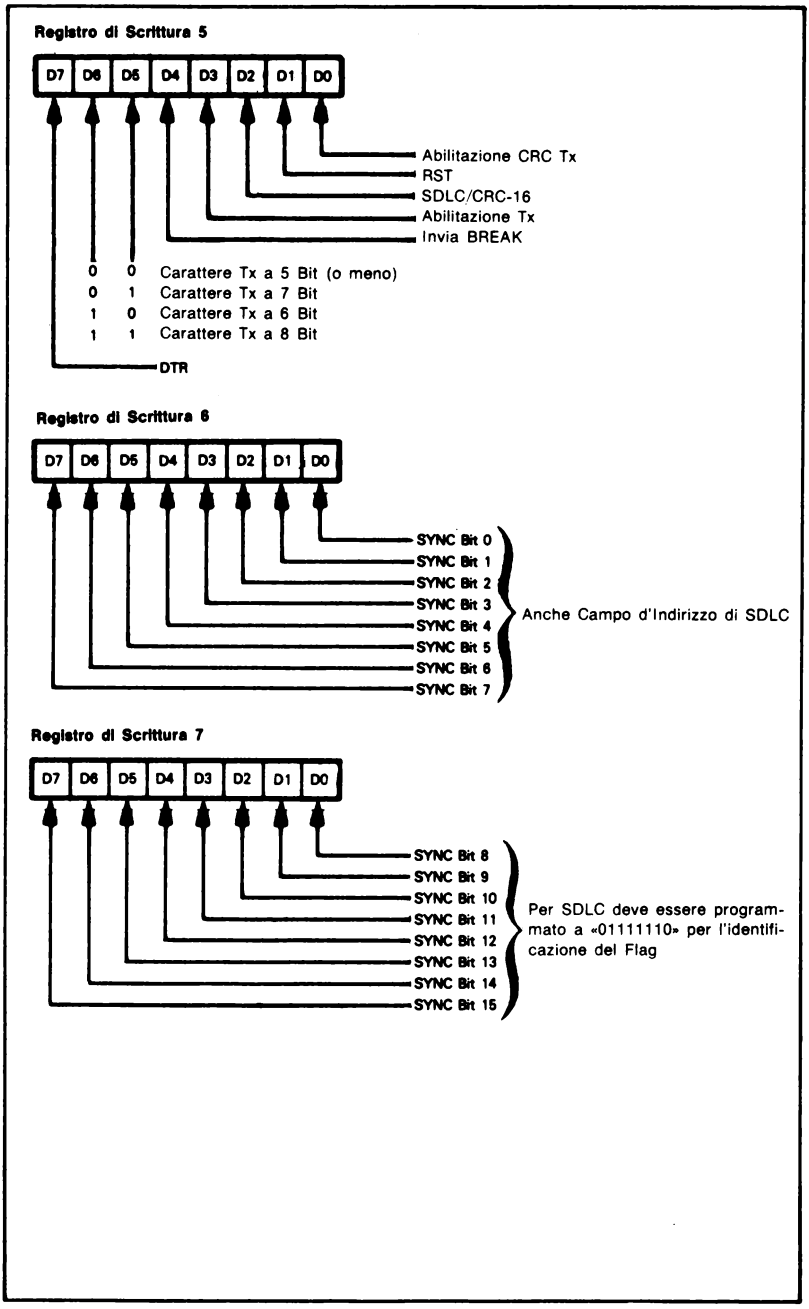


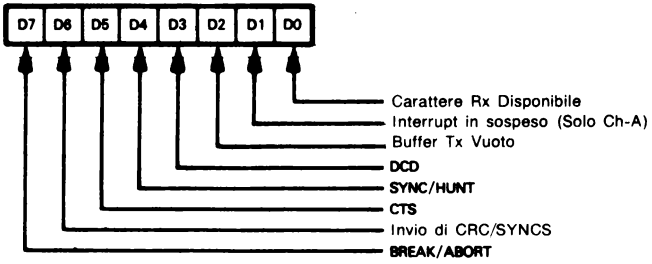
Figura 11-33. Controllo del SIO o Registri di Scrittura (continua)

Registri di Lettura

Il SIO Z80 contiene tre registri che possono essere letti per ottenere lo stato di ogni canale. Le informazioni di Stato comprendono condizioni di errore, vettore di interrupt, e segnali di protocollo di interfaccia di comunicazione standard. Per leggere il contenuto di un Registro di lettura scelto, il software del sistema deve scrivere nel SIO il byte contenente l'informazione del puntatore (D0-D2) esattamente nello stesso modo di una operazione su un registro di scrittura. Quindi, mediante un'operazione di LETTURA, si può leggere da CPU il contenuto del registro di Lettura 'Stato indirizzato.

La reale potenza di questo tipo di struttura di comando è che il programmatore ha completa libertà, dopo aver puntato il registro selezionato, sia di leggere o di scrivere per inizializzare o controllare quel registro. Progettando il software per inizializzare il SIO Z80 in modo modulare e strutturato, il programmatore può usare le potenti istruzioni di I/O di Blocco dello Z80 per semplificare in modo significativo e per accelerare il suo sviluppo e debug del software.

Registro di Lettura 0



Registro di Lettura 1

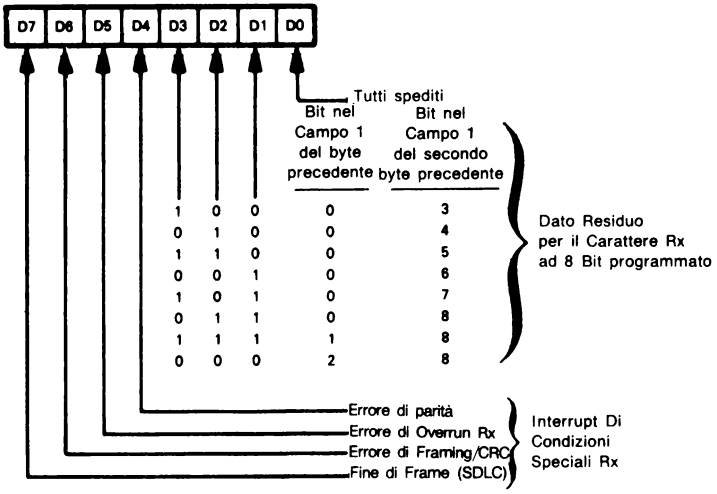


Figura 11-34. Stato del SIO o Registri di Lettura

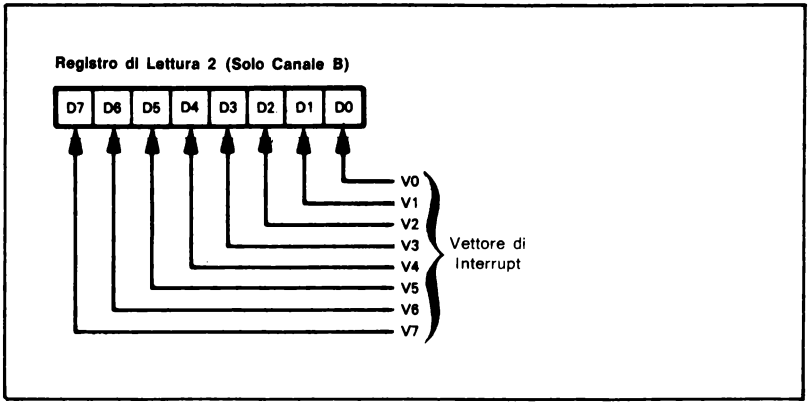


Figura 11-34. Stato del SIO o Registri di Lettura (continua)

Si notino le seguenti caratteristiche speciali del SIO:

CARATTERISTICHE SPECIALI DEL SIO

1) Le istruzioni d'ingresso e d'uscita indirizzano registri fisicamente distinti. Non c'è nessun modo per leggere i registri di controllo o per scrivere nei registri di stato.

2) Tutti i registri di controllo di un canale fanno capo ad un singolo indirizzo di porta. In tale modo si richiedono due byte per cambiare il contenuto di un registro di controllo tranne che per il Registro 0.

RESET DEL SIO

3) Il segnale $\overline{\text{RESET}}$ inizializza il SIO e scrive nel Registro 0. Disabilita pure i ricevitori ed i trasmettitori, disattiva tutti i segnali di controllo e disabilita tutti gli interrupt. Tratteremo il sistema d'interrupt del SIO nel Capitolo 12.

4) Il SIO deve essere configurato prima del suo utilizzo. Il modo più facile per fare ciò è di porre i byte richiesti in una tabella e di usare l'istruzione di I/O di Blocco ripetuta. La tabella deve comprendere entrambi i byte necessari per indirizzare i vari registri ed i dati da mettere in essi. Una tipica routine potrebbe essere:

```
LD      B,LENG      ;NUMERO DI PAROLE NELLA TABELLA
LD      C,SIOCRA    ;NUMERO DELLA PORTA
LD      HL,CTLTAB   ;PARTENZA DELLA TABELLA DI CONTROLLO
OTIR                ;CONFIGURA IL SIO
```

5) I segnali dell'RS-232 sono attivi bassi. Tuttavia i bit di controllo del SIO per questi segnali sono attivi alti (cioè un «1» logico in un bit di controllo trasmette un segnale RS-232 basso).

6) Il SIO richiede un clock esterno. In comunicazioni asincrone a 110 Baud, di solito si forniscono 1760 Hz e si usa il modo X16. Il SIO campionerà i bit alla frequenza del clock per motivi di sincronizzazione e per evitare Falsi bit di start provocati da rumore sulla linea.

7) Il flag Data Ready (Rx Character Available) è il bit 0 del Read Register 0. Il Peripheral Ready flag (Tx Buffer Empty) è il bit 2 del Read Register 0.

8) I bit di stato di errore (parità, overrun e framing) sono nel Read Register 1.

ESEMPI

I/O di una telescrivente mediante un USART

Compito 1: Leggere da una telescrivente mediante un SIO.

Scopo: Ricevere dati da una telescrivente mediante un SIO e mettere il dato nella locazione di memoria 0040. Il dato è in codice ASCII a 7 bit con parità dispari.

Programma Sorgente:

```
        LD      A,4             ;ACCEDI AL REGISTRO DI SCRITTURA 4
        OUT     (SIOCRA),A
        LD      A,01000001B     ;MODO DI CLOCK X 16, PARITÀ DISPARI
        OUT     (SIOCRA),A
        LD      A,3             ;ACCEDI AL REGISTRO DI SCRITTURA 3
        OUT     (SIOCRA),A
        LD      A,01000001B     ;CARATTERI A 7 BIT, ABILITA IL RICEVITORE
        OUT     (SIOCRA),A
        SUB     A               ;ACCEDI AL REGISTRO DI LETTURA 0
        OUT     (SIOCRA),A
WAITD:  IN      A,(SIOCRA)      ;PRENDI LO STATO
        RRA                     ;IL DATO È DISPONIBILE?
        JR      NC,WAITD        ;NO, ATTENDI
        IN      A,(SIODRA)      ;SÌ, PRENDI IL DATO
        LD      (40H),A         ;SALVA IL DATO IN MEMORIA
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Operativo)		
0000	3E		LD	A,4
0001	04			
0002	D3		OUT	(SIOCRA),A
0003	SIOCRA			
0004	3E		LD	A,01000001B
0005	41			
0006	D3		OUT	(SIOCRA),A
0007	SIOCRA			
0008	3E		LD	A,3
0009	03			
000A	D3		OUT	(SIOCRA),A
000B	SIOCRA			
000C	3E		LD	A,01000001B
000D	41			
000E	D3		OUT	(SIOCRA),A
000F	SIOCRA			
0010	97		SUB	A
0011	D3		OUT	(SIOCRA),A
0012	SIOCRA			
0013	DB	WAITD:	IN	A,(SIOCRA)
0014	SIOCRA			
0015	1F		RRA	
0016	30		JR	NC,WAITD
0017	FB			
0018	DB		IN	A,(SIODRA)
0019	SIODRA			
001A	32		LD	(40H),A
001B	40			
001C	00			
001D	76		HALT	

Il programma struttura il Write Register 4 come segue:

ESEMPIO DI CONFIGURAZIONE DEL SIO

Bit 7 e 6 = 01 per selezionare il modo di clock X16 (si devono fornire 1760 Hz)
Vit 1 = 0 per scegliere la parità dispari
Bit 0 = 1 per abilitare il controllo di parità.

Il programma struttura il Write Register 3 come segue:

Bit 7 e 6 = 01 per un carattere a 7 bit
Bit 0 = 1 per abilitare il ricevitore

Il bit di stato di dato ricevuto è il bit 0 del Read Register 0.

Si noti che ogni errore trovato sarà riportato nel Read Register 1.

STATO DI ERRORE DEL SIO

Bit 6 = per un errore di framing (nessun bit di stop)
Bit 5 = 1 per un errore di overrun
(più di un dato ricevuto prima della lettura del dato precedente)
Bit 4 = 1 per un errore di parità

Provate ad aggiungere una routine di controllo di errore al programma. Posizionare:

(0061) = 0 se non si sono verificati errori
= 1 se si è verificato un errore di parità
= 2 se si è verificato un errore di overrun
= 3 se si è verificato un errore di framing

Si noti che il ricevitore controlla sempre un solo bit di stop.

Compito 2: Scrivere nella telescrivente mediante un SIO

Scopo: Trasmettere un dato dalla locazione di memoria 0040 a una telescrivente mediante un SIO. Il dato è un codice ASCII a 7 bit con parità dispari.

Programma Sorgente:

```
        LD      A,4             ;ACCEDI AL REGISTRO DI SCRITTURA 4
        OUT     (SIOCRA),A
        LD      A,01001101B     ;MODO DI CLOCK X 16, 2 BIT DI STOP,
                                ;   PARITÀ DISPARI
        OUT     (SIOCRA),A
        LD      A,5             ;ACCEDI AL REGISTRO DI SCRITTURA 5
        OUT     (SIOCRA),A
        LD      A,00101000B     ;CARATTERI A 7 BIT, ABILITA IL TRASMETTITORE
        OUT     (SIOCRA),A
        SUB     A               ;ACCEDI AL REGISTRO DI LETTURA 0
        OUT     (SIOCRA),A
WAITR:  IN      A,(SIOCRA)      ;PRENDI LO STATO
        BIT     2,A             ;IL TRASMETTITORE È PRONTO?
        JR      Z,WAITR         ;NO, ATTENDI
        LD      A,(40H)         ;SÌ, PRENDI IL DATO
        OUT     (SIODRA),A      ;E TRASMETTILO
        HALT
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)		Istruzione (Codice Operativo)	
0000	3E		LD	A,4
0001	04			
0002	D3		OUT	(SIOCRA),A
0003	SIOCRA			
0004	3E		LD	A,01001101B
0005	4D			
0006	D3		OUT	(SIOCRA),A
0007	SIOCRA			
0008	3E		LD	A,5
0009	05			
000A	D3		OUT	(SIOCRA),A
000B	SIOCRA			
000C	3E		LD	A,00101000B
000D	28			
000E	D3		OUT	(SIOCRA),A
000F	SIOCRA			
0010	97		SUB	A
0011	D3		OUT	(SIOCRA),A
0012	SIOCRA			
0013	DB	WAITR:	IN	A,(SIOCRA)
0014	SIOCRA			
0015	CB		BIT	2,A
0016	57			
0017	28		JR	Z,WAITR
0018	FA			
0019	3A		LD	A,(40H)
001A	40			
001B	00			
001C	D3		OUT	(SIODRA),A
001D	SIODRA			
001E	76		HALT	

Il programma struttura il Write Register 4 come segue:

Bit 7 e 6 = 01 per selezionare il modo di clock X 16
(si devono fornire 1760 Hz)
Bit 3 e 2 = 11 per aggiungere 2 bit di stop ad ogni carattere
Bit 1 = 0 per scegliere una parità dispari
Bit 0 = 1 per abilitare la generazione di parità

Il programma struttura il Write Register 5 come segue:

Bit 6 e 5 = 01 per caratteri a 7 bit
Bit 3 = 1 per abilitare il trasmettitore

Il bit di stato del trasmettitore è il bit 2 del Read Register 1.

INTERFACCE STANDARD

Per collegare periferiche al microcomputer si possono usare altre interfacce standard oltre all'RS-232 e al TTY current-loop. Tra i più diffusi si ha:

INTERFACCE STANDARD

1) Le interfacce seriali RS449, RS422 e RS423[13].
2) Il General Purpose Interface Bus parallelo ad 8 bit, noto pure come IEEE-488 o come Hewlett-Packard Interface Bus (HPIB)[14]
3) Il S-100 o bus per hobbisti Altair/Imsai.[15] Pure questo è un bus ad 8 bit.
4) L'Intel Multibus.[16] Questo è un altro bus ad 8 bit che può, tuttavia, essere espanso a gestire 16 bit in parallelo.

PROBLEMI

1) Separazione delle chiusure da una Tastiera Non codificata

Scopo: Il programma dovrebbe leggere gli ingressi da una tastiera 3 x 3 non codificata e porli in un vettore. Il numero di ingresso richiesto è nella locazione di memoria 0040 e il vettore inizia nella locazione di memoria 0041.

Separate una chiusura dalla successiva attendendo la fine della corrente di chiusura. Ricordatevi di eliminare i rimbalzi della tastiera (questo può essere fatto semplicemente con un'attesa di 1 ms).

Problema Campione:

```
            (0040) = 04
            Gli ingressi sono 7, 2, 2, 4
Risultato:  (0041) = 07
            (0042) = 02
            (0043) = 02
            (0044) = 04
```

2) Leggere una Frase da una Tastiera Codificata

Scopo: Il programma dovrebbe leggere gli ingressi da una tastiera in ASCII (7 bit con bit di Parità Zero) e metterli in un vettore fino alla ricezione di un periodo in ASCII (2E esadecimale). L'array parte dalla locazione di memoria 0040. Ogni ingresso è indicato da un segnale di strobe come nell'esempio dato come Una Tastiera Codificata.

Problema Campione:

```
            Gli ingressi sono: H, E, L, L, O.
Risultato:  (0040) = 48  H
            (0041) = 45  E
            (0042) = 4C  L
            (0043) = 4C  L
            (0044) = 4F  0
            (0045) = 2E  .
```

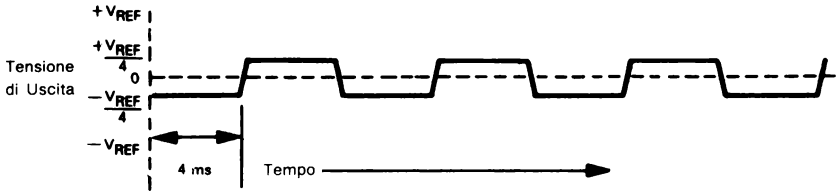
3) Generatore d'Onda Quadra di Ampiezza Variabile

Scopo: Il programma dovrebbe generare un'onda quadra, come indicato nella Figura successiva, usando un convertitore D/A. La locazione di memoria 0040 contiene l'ampiezza scalare dell'onda, la locazione di memoria 0041 la lunghezza di un mezzo ciclo in millisecondi, e la locazione di memoria 0042 il numero di cicli.

Supponiamo che un'uscita digitale pari a 80_{16} verso il convertitore dia come risultato un'uscita analogica di zero volt. In generale un'uscita digitale pari a D darà come risultato un'uscita analogica pari a (D—80)/80 X -VREF volt.

Problema Campione:

(0040) = A0 (esadecimale)
(0041) = 04
(0042) = 03
Risultato:



La tensione di base è 80_{16} = 0 volt
Il fondo scala è 100_{16} = -VREF/volt
Quindi $A0_{16}$ = (A0—80)/80 X -VREF = -VREF/4

Il programma produce 3 impulsi di ampiezza VREF/4 con metà ciclo lungo 4 ms.

4) Media di Letture Analogiche

Scopo: Il programma dovrebbe prendere quattro letture da un convertitore A/D ogni dieci millisecondi e mettere la media nella locazione di memoria 0040. Supponiamo che si ignori il tempo di conversione A/D.

Problema Campione:

Le letture sono (esadecimali) 86, 89, 81, 84
Risultato: (0040) = 85

5) Terminale a 30 Caratteri al Secondo

Scopo: Modificare le routine di trasmissione e di ricezione dell'esempio dato come Una telescrivente che gestisce un terminale a 30 cps che trasferisce dati in codice ASCII con un solo bit di stop e con parità pari. Come si potrebbero scrivere i programmi per gestire un terminale o l'altro in funzione di un bit di flag nella locazione di memoria 0060; per esempio (0060) = 0 per il terminale a 30 cps, (0060) = 1 per il terminale a 10 cps?

BIBLIOGRAFIA

1. Barnes, J., e V. Gregory, "Use Microcomputers to Enhance Performance with Noisy Data", EDN, August 20, 1976, pp. 71-72.

2. Swanson, R., "Understanding Cyclic Redundancy Codes". Computer Design, November 1975, pp. 93-99; and McNamara, J. E., Technical Aspects of Data Communication, Digital Equipment Corp., Maynard, Mass. 1977.

3. Per esempio il Controllore di Accesso Diretto in Memoria (o DMA) per microcomputer basati sullo Z80 è descritto in An Introduction to Microcomputers: Volume 2 – Some Real Microprocessors.

4. The TTL Data Book for Design Engineers, Texas Instruments Inc., P. O. Box 5012. Dallas, Texas 75222, 1976.

5. Dilatush, E., "Special Report: Numeric and Alphanumeric Displays", EDN, February 5, 1978, pp. 26-35.

6. Vedere il Riferimento 4.

7. Hnatek, E. R., A User's Handbook of D/A and A/D Converters, Wiley, New York, 1976.

8. Vedere il Riferimento 7.

9. Vedere pure D. Guzeman, "Marry Your µP to Monolithic A/Ds"; Electronic Design, January, 18, 1977, pp. 82-86.

10. Per una trattazione sull'UART vedere P. Rony et al., "The Bugbook II[a]," E and L Instruments Inc, 61 First Street, Derby, CT. 06418; o D. G. Larsen et al., "INWAS: Interfacing with Asynchronous Serial Mode", IEEE Transactions on Industrial Electronics and Control Instrumentation, February 1977, pp. 2-12. Vedere pure McNamara, Riferimento 2.

11. Lo standard ufficiale di RS-232 è disponibile come "Interface between Data Terminal Equipment and Data Communications Equipment Employing Serial Binary Data Interchange", EIA RS-232C August, 1969. Potete trovare descrizioni introduttive dell'RS-232 in G. Pickles, "Who's Afraid of RS-232?", Kilobaud, May 1977, pp. 50-54 e in C.A. Ogdin, "Microcomputer Buses – Parte II", Mini-Micro Systems, July 1978, pp. 76-80. Ogdin descrive il più recente RS-449 standard.

12. Il SIO è pure trattato molto completamente nel Volume 3 di An Introduction to Microcomputers; i seguenti riferimenti bibliografici descrivono il suo uso come controllore di link di dati: Weissberger, A.J., "Data-Link Control Chips; Bringing Order to New Protocols", Electronics June 8, 1978, pp. 104-112.

13. Electronic Industries Association, "Electrical Characteristics of Balanced Voltage Digital Interface Circuits," EIA RS-422, April 1975.
Electronic Industries Association, "Electrical Characteristics of Unbalanced Voltage Digital Interface Circuits,", EIA RS-423, April 1975.
Electronic Industries Association, "General Purpose 37-Position and 9-Position Interface for Data Terminal Equipment and Data Circuit Terminating Equipment Employing Serial Binary Data Interchange," EIA RS-449, November 1977.
Morris, D., "Revised Data Interface Standards,", Electronic Design, September 1, 1977, pp. 138-141.

14. Institute of Electrical and Electronics Engineers, "IEEE Standard Digital Interface for Programmable Instrumentation," IEEE Std 488-1975-ANSI MC 1.1-1975.
J. B. Peatman, Microcomputer-Based Designg McGraw-Hill, New York, 1977; Loughry, D. C. and M. S. Allen, "IEEE Standard 488 and Microprocessor Synergism," Proceedings of the IEEE, February 1978, pp. 162-172.

15. Morrow, G., and H. Fullmer, "Proposed Standard for the S-100 Bus," Computer, May 1978, pp. 84-89.
Smith, M. L., "Build Your Own Interface", Kilobaud, June 1977, pp. 22-28.

16. Rolander, T., "Intel Multibus Interfacing," Intel Application Note AP-28, Intel Corporation, Santa Clara, CA., 1977.

Capitolo 12

INTERRUPT

Gli interrupt sono degli ingressi che la CPU prende in esame come parte di ogni ciclo d'istruzione. Questi ingressi permettono alla CPU di rispondere ad eventi esterni in una maniera più efficace di ciascun dispositivo che funzioni a scansione. Quando si utilizzano gli interrupt per inizializzare l'I/O, generalmente con più hardware del consueto, è necessario un I/O programmato, ma ciò fornisce una risposta più veloce ed immediata.[1]

RAGIONAMENTO DIETRO GLI INTERRUPT

Perché utilizzare gli interrupt? Essi permettono la cattura dell'attenzione immediata della CPU da parte di eventi quali gli allarmi, la mancanza di tensione di alimentazione, il trascorrere di una certa quantità di tempo, e da parte di periferiche che hanno dei dati o sono pronte ad accettarli. Il programmatore non ha necessità di scandire ogni dispositivo e neppure si deve preoccupare che il sistema perda di vista completamente gli avvenimenti esterni. Un sistema d'interrupt è come il campanello di un telefono — esso suona quando si riceve una chiamata di modo che non si debba risollevare occasionalmente il ricevitore per vedere se qualcuno è in linea. La CPU può abbandonare il suo normale lavoro (e far eseguire molte più cose). Quando avviene qualcosa, l'interrupt stimola la CPU e la obbliga a servire l'ingresso prima di riprendere le normali operazioni. Naturalmente, questa semplice descrizione diventa più complicata (esattamente come un centralino telefonico) quando vi sono molti interrupt di diversa importanza e vi sono compiti che non possono essere interrotti.

CARATTERISTICHE DEI SISTEMI D'INTERRUPT

L'implementazione di sistemi funzionanti ad interrupt varia grandemente. Tra le domande che devono trovare una risposta per caratterizzare un sistema particolare vi sono:

1) Quanti ingressi d'interrupt ci sono?
2) Come risponde la CPU ad un interrupt?
3) La CPU, come determina la sorgente di un interrupt se il numero delle sorgenti supera il numero degli ingressi?
4) La CPU, può discriminare fra interrupt importanti e irrilevanti?
5) Come e quando il sistema viene abilitato e disabilitato all'interrupt?

Vi sono molte risposte differenti a queste domande. Lo scopo di tutte le implementazioni, tuttavia, è che la CPU risponda celermente agli interrupt ed in seguito riprenda la sua normale attività.

Il numero degli ingressi d'interrupt nel chip della CPU determina il numero di differenti risposte che la CPU può produrre senza alcuna aggiunta di hardware oppure software. Ciascun ingresso può produrre una risposta interna differente. Purtroppo, la maggior parte dei microprocessori posseggono un numero molto basso (tipicamente uno o due) di ingressi separati d'interrupt.

La risposta fondamentale della CPU ad un interrupt deve essere il trasferimento del controllo alla «routine» corretta di servizio dell'interrupt ed il salvataggio del valore corrente del Contato-

re di Programma. La CPU perciò deve eseguire un Salto-alla-Subroutine o una istruzione Call che ha come indirizzo l'inizio della «routine» di servizio dell'interrupt. Quest'azione salverà l'indirizzo di ritorno nello Stack e trasferirà il controllo alla routine di servizio dell'interrupt. La quantità di hardware esterno richiesto per produrre questa risposta varia moltissimo. Alcune CPU generano internamente l'istruzione e l'indirizzo; altre necessitano di hardware esterno per formarli. La CPU può solo generare un'istruzione o un indirizzo differente per ciascun ingresso separato.

POLLING
VETTORIZZAZIONE

Se il numero dei dispositivi interrompenti supera il numero degli ingressi, la CPU avrà necessità di hardware o software per identificare la sorgente dell'interrupt. Nel caso più semplice, il software può essere una routine di scansione che controlla lo stato dei dispositivi che potrebbero aver interrotto. Il solo vantaggio di tale sistema rispetto alla normale scansione è che la CPU è a conoscenza che almeno un dispositivo è attivo. La soluzione alternativa prevede l'uso di hardware supplementare per fornire un unico dato d'ingresso (o «vettore») per ciascuna sorgente. Le due alternative possono essere combinate insieme; i vettori possono identificare gruppi d'ingressi dai quali la CPU può a sua volta identificarne uno in particolare, mediante scansione.

PRIORITÀ

Un sistema d'interrupt che può eseguire una discriminazione tra interrupt importanti e irrilevanti viene chiamato «sistema d'interrupt con priorità». Un hardware interno può fornire tanti livelli di priorità quanti sono gli ingressi. Un hardware esterno può inoltre fornire livelli supplementari mediante l'uso di un registro di Priorità e un comparatore. L'hardware esterno non concede all'interrupt di raggiungere la CPU a meno che la sua priorità sia più alta del contenuto del registro di Priorità. Un sistema d'interrupt con priorità può aver bisogno di un modo particolare di trattamento degli interrupt a bassa priorità che possono essere ignorati per lunghi periodi di tempo.

ABILITAZIONE E DISABILITAZIONE DEGLI INTERRUPT

La maggior parte dei sistemi funzionanti ad interrupt possono essere abilitati o disabilitati. Infatti, la maggior parte delle CPU disabilitano gli interrupt allorquando viene compiuta una operazione di RESET (in modo tale che il programmatore possa configurare il sistema d'interrupt) e nell'accettare un interrupt (di modo che l'interrupt non interromperà la propria routine di servizio). Il programmatore può desiderare la disabilitazione degli interrupt mentre si stanno preparando od elaborando dei dati, compiendo un ciclo di temporizzazione, oppure eseguendo un'operazione multibyte.

INTERRUPT NON MASCHERABILE

Un interrupt che non può essere disabilitato (talvolta chiamato «interrupt non mascherabile») può essere utile per avvisare della mancanza di tensione d'alimentazione, un evento che naturalmente deve avere la precedenza su tutte le altre possibili attività.

SVANTAGGI DEGLI INTERRUPT

I vantaggi degli interrupt sono ovvi, ma esistono anche degli svantaggi. Questi comprendono:

1) I sistemi d'interrupt possono richiedere una grande quantità di hardware supplementare.

2) Gli interrupt esigono che i trasferimenti di dati avvengano ancora sotto il controllo del programma tramite la CPU. Non c'è alcun vantaggio dal punto di vista della velocità come con il DMA.

3) Gli interrupt sono ingressi casuali, che rendono difficoltose le operazioni di debugging e di collaudo. Possono accadere occasionalmente degli errori, e pertanto possono essere molto difficili da scoprire.[2]

4) Gli interrupt possono comportare una grande quantità di supervisione se devono essere salvati molti registri e se deve esserne determinata la sorgente, mediante scansione.

SISTEMA D'INTERRUPT DELLO Z80

La risposta interna ad un interrupt da parte dello Z80 è abbastanza complessa, dato che vi sono tre diversi modi di funzionamento. Il sistema d'interrupt consiste di:

INGRESSI DI INTERRUPT DELLO Z80

1) Un ingresso d'interrupt mascherabile attivo — basso ($\overline{INT}$) e un ingresso d'interruzione non mascherabile anch'esso attivo — basso ($\overline{NMI}$).
2) Due flip-flop d'abilitazione (IFF1 e IFF2). IFF1 può essere posizionato ad 1 oppure azzerato per abilitare o disabilitare gli eventuali interrupt. IFF2 serve come memorizzazione temporanea di IFF1 durante gli interrupt non mascherabili.

RISPOSTA AGLI INTERRUPT DELLO Z80

Lo Z80 verifica lo stato corrente del sistema d'interrupt al termine di ciascun ciclo di istruzione. Se un interrupt risulta attivo ed abilitato, la risposta è la seguente:[3]

1) La CPU disabilita il sistema d'interrupt azzerando IFF1. IFF2, comunque, viene lasciato nel suo stato originale nel caso sia avvenuta un interrupt di tipo non mascherabile. Notate che $\overline{RESET}$ azzera ambedue i flip-flop dell'interrupt in modo tale che il sistema possa essere configurato prima che vengano abilitati gli interrupt.
2) La CPU esegue un particolare ciclo di Riconoscimento dell'Interrupt, distinto dal segnale $\overline{M1}$ (prelievo del codice operativo o «fetch») che è attivo, da $\overline{MREQ}$ (richiesta in memoria) che è inattivo (così la CPU non eseguirà il suo normale accesso alla memoria), e da IORQ (richiesta d'input/output) che è attivo in modo che possa essere collocato sul Bus Dati un vettore di risposta all'interrupt.

La parte restante della risposta dipende dal modo d'interrupt e dalla sorgente.

Notate in particolare che lo Z80 controllerà gli interrupt dopo ogni trasferimento o paragone in un'istruzione I/O di Spostamento di Blocco, Confronto di Blocco, o di Blocco Ripetuta.

ISTRUZIONE DI INTERRUPT DELLO Z80

Lo Z80 ha le seguenti istruzioni particolari per l'uso con il sistema d'interrupt:

1) EI (Abilitazione agli Interrupt) abilita l'interrupt mascherabile posizionando ad 1 i flip-flop d'interrupt.
2) DI (Disabilitazione degli Interrupt) disabilita l'interrupt mascherabile azzerando i flip-flop d'interrupt.
3) RST («Restart») è un'istruzione «Call» ad un solo byte che salva il contenuto corrente del Contatore del Programma nello Stack e salta all'indirizzo specificato nell'istruzione. La Tabella 12-1 contiene le varie istruzioni di «Restart» ed i loro indirizzi di destinazione. RST viene sovente utilizzata nei sistemi d'interrupt perchè è una istruzione di una sola parola che è facile da comporre e da collocare sul Bus Dati.
4) RETI (Ritorno dall'Interrupt) agisce esattamente come una normale istruzione di Ritorno (RET) tranne che i dispositivi periferici dello Z80 (i PIO, SIO, e CTC) riconoscono quest'istruzione e la usano come una notificazione che la «routine» corrente di servizio dell'interrupt è stata completata.
5) RETN (Ritorno dall'Interrupt Non Mascherabile) agisce esattamente come una normale istruzione di Ritorno (RET) tranne che essa carica IFF1 da IFF2 in modo da ripristinare lo stato originale del sistema d'interrupt.
6) LD A,I carica l'Accumulatore con il contenuto del Registro I (Vettore d'Interrupt). Questa istruzione (e LD A,R) inoltre pone IFF2 nel bit P/O del registro dei Flag. Quel flag può quindi essere esaminato oppure salvato nello Stack.
7) LD I, A carica il registro I (Vettore d'Interrupt) con il contenuto dell'Acccumulatore.
8) IM (Posizionamento del Modo d'Interrupt) determina il modo nel quale vengono serviti gli interrupt. Le tre opzioni sono 0,1 oppure 2; queste verranno successivamente descritte in questo capitolo.

Interrupt Non Mascherabile

INTERRUPT NON MASCHERABILE DELLO Z80

L'interrupt non mascherabile è un ingresso sensibile al fronte (innescato dal fronte negativo). Il processore quindi reagisce solamente al fronte di un impulso su questa linea, e l'impulso non interromperà la propria routine di servizio. Gli interrupt non mascherabili son utili per applicazioni che devono rispondere a una perdita di potenza (cioè, devono salvare i dati in una memoria a basso consumo oppure eseguire una commutazione verso una batteria di «backup». Le applicazioni tipiche sono apparecchiature per mezzi di comunicazione che debbono conservare dei codici e dei messaggi parziali ed apparecchiature di prova che debbono eseguire delle prove parzialmente completate.

Tabella 12-1. L'istruzione Restart (RST)

Istruzione RST (Codice Mnemonico)	Codice Operativo (Esadec)	Indirizzo di Destinazione (Esadec)	(Decimale)
RST 0	C7	0000	0
RST 8	CF	0008	08
RST 10H	D7	0010	16
RST 18H	DF	0018	24
RST 20H	E7	0020	32
RST 28H	EF	0028	40
RST 30H	F7	0030	48
RST 38H	FF	0038	56

Lo Z80 risponde ad un interrupt non mascherabile come segue:

1) Azzera IFF1, disabilitando in tal modo tutti gli interrupt (ma conservando il vecchio stato di IFF1 dentro IFF2).
2) Non considera la successiva istruzione prelevata dalla memoria ed invece salta alla locazione di memoria 0066_{16}, salvando il vecchio valore del Contatore del Programma nello Stack.

Ricordate che al termine della routine di servizio un'istruzione RETN ripristinerà il vecchio stato di IFF1 da IFF2.

Non verrà ulteriormente trattato l'interrupt non mascherabile. D'ora in poi, si supporrà che tutti gli ingressi interrompenti siano collegati ad $\overline{INT}$.

Modi d'Interrupt dello Z80

MODI DI INTERRUPT

Lo Z80 ha tre modi d'interrupt. Il programmatore può scegliere uno qualsiasi di questi modi mediante l'appropriata istruzione IM. Con l'operazione di «reset», il processore entra sempre nel Modo 0. I modi sono:

Modo 0

In questo modo, la CPU utilizza il dato d'ingresso, durante il ciclo di Riconoscimento dell'Interrupt, come fosse un'istruzione. Questo modo è analogo al modo di risposta all'interrupt dell'8080[4].

Il dato normale d'ingresso che deve essere fornito dall'esterno è un'istruzione RST (vedi Tabella 12-1).

RST è utile in un sistema d'interrupt per i seguenti motivi:

ISTRUZIONE RESTART

1) È una istruzione di una parola e quindi richiede solamente un ciclo di raccolta o fetch.
2) Fornisce otto diversi indirizzi di destinazione o vettori.
3) I suoi vettori sono sufficientemente distanziati uno dall'altro da permettere istruzioni di Salto onde raggiungere le reali routine di servizio.
4) È facile da costruire, dato che cinque dei suoi bit sono sempre «1». Un codificatore da 8 a 3 linee può fornire piuttosto semplicemente gli altri tre bit.

RST ha i seguenti svantaggi:

1) Non può fornire più di otto vettori.
2) I suoi vettori non sono sufficientemente distanziati uno dall'altro da concedere dello spazio per le intere routine di servizio degli interrupt.
3) I suoi vettori sono contenuti in un'area di memoria.
4) RST 0 ha lo stesso indirizzo di destinazione dell'ingresso $\overline{\text{RESET}}$ ed è quindi piuttosto difficoltoso da usare. Il sistema ha bisogno di hardware per eseguire una discriminazione tra $\overline{\text{RESET}}$ e $\overline{\text{RST}}$ 0, dato che i due non si possono distinguere col solo software.

Ricordate che RST salva il vecchio Contatore del Programma nello Stack proprio come fa l'istruzione CALL.

Modo 1

In questo modo, la CPU non considera il dato d'ingresso durante il ciclo di Riconoscimento dell'Interrupt ed esegue sempre RST 38H, saltando così alla locazione 0038_{16} e salvando il vecchio contenuto del Contatore del Programma nello Stack. Questo modo è equivalente al Modo 0 se il dato d'ingresso è sempre RST 38H (FF_{16}).

Il vantaggio di questo modo è che non viene richiesto dell'hardware esterno. I suoi inconvenienti sono che non esiste alcun modo di eseguire una diversificazione immediata tra le varie sorgenti dell'interrupt e che l'indirizzo di destinazione è fissato. Il Modo 1 è comodo in applicazioni che hanno solo una o due sorgenti per l'interrupt e nelle quali è essenziale ridurre il costo dell'hardware.

Modo 2

In questo modo, la CPU usa il dato d'ingresso come porzione di un indirizzo dal quale poi ottenere l'indirizzo di partenza della routine di servizio dell'interrupt. Quando viene accettata un interrupt, la CPU:

1) Disabilita gli ulteriori interrupt azzerando IFF1 e IFF2.
2) Memorizza il vecchio contenuto del Contatore del Programma nello Stack RAM.
3) Costruisce un puntatore col contenuto del Registro I (gli otto MSB) e dell'ingresso presente sul Bus Dati durante il ciclo di Riconoscimento dell'Interrupt (gli otto LSB). Il bit meno significativo di questo puntatore viene forzato a zero.
4) Preleva un indirizzo dalle due locazioni di memoria cominciando da quello indicato dal puntatore (vedi Figura 12-1).
5) Trasferisce il controllo all'indirizzo ricavato dalla memoria.

La risposta all'interrupt in questo modo necessita di 19 cicli di clock.

Il vantaggio di questo modo è che esso può fornire una pagina intera di 128 vettori di servizio dell'interrupt situati in qualsiasi punto della memoria. Gli svantaggi di questo modo sono che la risposta all'interrupt è più lenta e che il sistema deve essere inizializzato, come segue:

1) La tabella dei vettori deve essere caricata in memoria se non è presente in ROM.
2) Il registro I deve essere caricato con gli otto bit più o meno significativi (o numero di pagina) dell'indirizzo della tabella. Notate che l'operazione di $\overline{RESET}$ azzera il registro I. Si può caricare quest'ultimo con un valore nel modo seguente:

```
LD      A,IPGNO  ;RACCOLTA DEL NUMERO DI PAGINA DELL'INTERRUPT
LD      I,A      ;INCAMERAMENTO NEL REGISTRO DEL VETTORE
```

3) Il Modo d'Interrupt 2 deve essere predisposto con l'istruzione IM 2.

Il Modo 2 è destinato a funzionare con i PIO, SIO, e CTC dello Z80. Gli interrupt relativi al SIO e al PIO sono descritti successivamente in questo capitolo.

COMPATIBILITÀ DELL'INTERRUPT Z80/8080

Il Modo 0 per il sistema d'interrupt dello Z80 è, come è stato richiamato, identico alla risposta dell'8080 all'interrupt. L'8080 non possiede i Modi d'Interrupt 1 o 2, sebbene il Modo 1 sia proprio veramente un caso particolare del Modo 0. Inoltre l'8080 non ha un ingresso $\overline{NMI}$.

L'8085 possiede degli ingressi d'interrupt supplementari, non disponibili sull'8080 e neppure sullo Z80. L'8085 inoltre ha un interrupt non mascherabile (chiamato TRAP) che forza una chiamata ad un diverso indirizzo (24_{16}) rispetto a quello utilizzato dall'ingresso $\overline{NMI}$ dello Z80.

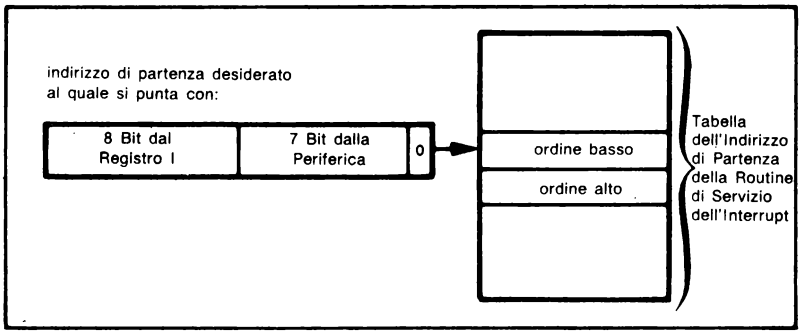


Figura 12-1. Generazione di un Vettore di Interrupt nel Modo di Interrupt 2.

INTERRUPT DEL PIO

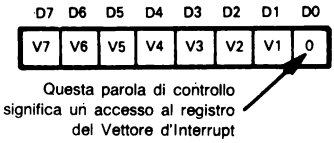
INTERRUPT DEL PIO

La maggior parte dei sistemi d'interrupt dello Z80 comprendono i PIO. Ciascuna porta del PIO ha le seguenti caratteristiche nell'uso con gli interrupt:

1) Un registro del Vettore d'Interrupt ad 8 bit, utilizzato per contenere gli otto bit meno significativi dell'indirizzo della tabella formato dalla CPU nel Modo d'Interrupt 2.
2) Un bit di abilitazione all'interrupt.
3) Un registro di Controllo dell'Interrupt per determinare l'operazione logica eseguita e la polarità attiva controllata per generare interrupt nel modo di controllo.
4) Un registro di Maschera dell'Interrupt usato per determinare quali linee di dati saranno controllate per generare degli interrupt nel modo di controllo.

VETTORE DI INTERRUPT DEL PIO

Si può accedere al registro del Vettore d'Interrupt in ogni porta scrivendo una parola di controllo con uno zero nel suo bit meno significativo, come indicato di seguito (vedi anche la Tabella 11-2):



significa che questa parola di controllo è un vettore d'interrupt.

Una sequenza tipica per stabilire il valore contenuto in questo registro è:

```
LD      A,IVECT
OUT     (PIOCR),A
```

dove IVECT ha uno «0» nel suo bit meno significativo. L'indirizzo di partenza per la routine di servizio dell'interrupt è all'indirizzo IVECT nella pagina assegnata alla tabella degli indirizzi per le routine di servizio.

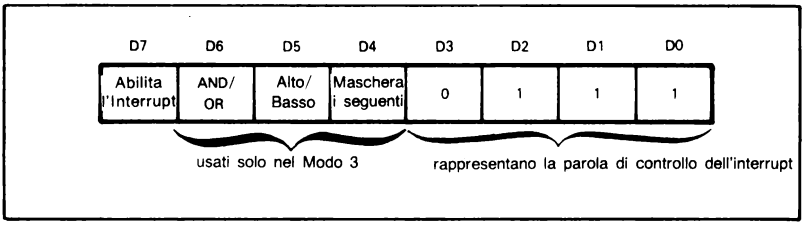


Figura 12-2. Formato di una Parola di Controllo Interrupt del PIO

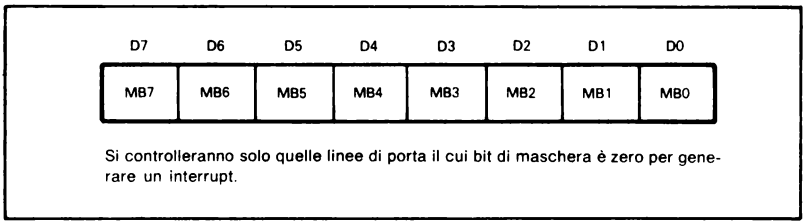


Figura 12-3. Formato di una Maschera di Interrupt del PIO

MODO DI CONTROLLO DI INTERRUPT DEL PIO

Si può collocare la parola di controllo dell'interrupt in ciascuna porta scrivendo una parola di controllo con il formato mostrato in Figura 12-2. Se la porta è nel Modo 3, i bit D6, D5, e D4 hanno i seguenti significati:

1) D6 = 1 significa tutte le linee I/O controllate debbono diventare attive per provocare un interrupt (cioè un AND logico), mentre D6 = 0 significa che qualsiasi linea I/O controllata che diventa attiva provocherà un interrupt (cioè un OR logico).

Notate che un interrupt avviene solamente se l'equazione logica è vera quando gli interrupt sono abilitati o se essa varia da falsa a vera mentre sono abilitati gli interrupt.

2) D5 definisce la polarità attiva (alto o basso) delle linee I/O controllate. D5 = 1 significa alto attivo. D5 = 0 significa basso attivo.
3) D4 = 1 significa che la successiva parola di controllo è una maschera d'interrupt (Figura 12-3). Solamente le linee con un bit di maschera pari a zero verranno controllate. D4 = 0 significa che non segue la maschera.

ABILITAZIONE E DISABILITAZIONE DEGLI INTERRUPT DEL PIO

Il bit 7 della parola di controllo dell'interrupt determina per la porta il valore del flip-flop d'abilitazione all'interrupt. Gli interrupt possono essere generati se il flip-flop si azzera, ma ricordate che il PIO non ha nessun ingresso di RESET. Il flip-flop di abilitazione all'interrupt può essere posizionato ad 1 od azzerato, senza influire sul resto della parola di controllo dell'interrupt, scrivendo una parola di controllo con il valore del flip-flop nel bit 7 e 0011 nei quattro bit meno significativi.

Il posizionamento ad 1 del bit 4 della parola di controllo dell'interrupt annulla qualsiasi interrupt in sospeso. Ciò può essere utilizzato per sospendere degli interrupt che possono essere capitati inavvertitamente durante un'operazione di reset.

Esempi

ESEMPI DI CONFIGURAZIONE DEGLI INTERRUPT DEL PIO

1) La porta dell'uscita interrompe con il vettore posizionato all'indirizzo 80_{16}. Ricordate che il numero delle pagine è contenuto nel registro I della CPU.

```
LD      A,00001111B   ;RENDI LA PORTA B UN'USCITA
OUT     (PIOCRB),A
LD      A,80H         ;INDIRIZZO DEL VETTORE = 80 ESADEC
OUT     (PIOCRA),A
LD      A,10000011B   ;ABILITA L'INTERRUPT DEL PIO
OUT     (PIOCRB),A
```

Un metodo alternativo che annulla gli interrupt in sospeso oltre ad abilitare gli interrupt dalla porta è il seguente:

```
LD      A,10010111B   ;ABILITA L'INTERRUPT DEL PIO
OUT     (PIOCRA),A
```

Un interrupt si verificherà in corrispondenza del fronte di salita di $\overline{STB}$.

2) La porta d'ingresso interrompe il vettore situato all'indirizzo 60_{16}.

```
LD      A,01001111B   ;RENDI LA PORTA A UN INGRESSO
OUT     (PIOCRA),A
LD      A,60H         ;INDIRIZZO DEL VETTORE = 60 ESADEC
OUT     (PIOCRA),A
LD      A,10000011B   ;ABILITA L'INTERRUPT DEL PIO
OUT     (PIOCRA),A
```

Un interrupt si verificherà in corrispondenza del fronte di salita di $\overline{STB}$.

3) La porta di controllo interrompe con il vettore situato all'indirizzo 48_{16}. Un interrupt verrà generato se le linee dei dati A_4 e A_7 si portano entrambe a basso.

```
LD      A,11001111B  ;RENDI LA PORTA A UN CONTROLLO
OUT     (PIOCRA),A
LD      A,10001000B  ;LE LINEE 4, 7 INGRESSI - LE ALTRE USCITE
OUT     (PIOCRA),A
LD      A,48H        ;INDIRIZZO DEL VETTORE = 48 ESADEC
OUT     (PIOCRA),A
LD      A,11010111B  ;ABILITA L'INTERRUPT DEL PIO
OUT     (PIOCRA),A
LD      A,01110111B  ;CONTROLLA SOLO LE LINEE 4, 7
OUT     (PIOCRA),A
```

La parola di controllo dell'interrupt ha:

il bit 7 = 1 per abilitare l'interrupt
il bit 6 = 1 per originare un interrupt solo se tutte le linee di controllo sono o divengono attive (un AND logico)
il bit 5 = 0 per specificare che uno «0» logico è lo stato attivo da controllare
il bit 4 = 1 per indicare che una parola di maschera viene di seguito (e per annullare gli interrupt in sospeso).

4) La porta di controllo che interrompe con il vettore posizionato all'indirizzo 28_{16}. Un interrupt verrà generato se una qualsiasi delle linee dei dati si porta ad alto.

```
LD      A,11001111B  ;RENDI LA PORTA B UN CONTROLLO
OUT     (PIOCRB),A
LD      A,0FFH       ;TUTTE LE LINEE INGRESSI
OUT     (PIOCRB),A
LD      A,28H        ;INDIRIZZO DEL VETTORE = 28 ESADEC
OUT     (PIOCRB),A
LD      A,10110111B  ;ABILITA GLI INTERRUPT
OUT     (PIOCRB),A
SUB     A            ;CONTROLLA TUTTE LE LINEE
OUT     (PIOCRB),A
```

La parola di controllo dell'interrupt ha:

il bit 7 = 1 per abilitare all'interrupt
il bit 6 = 0 per originare un interrupt se una qualsiasi delle linee controllate diventa attiva (un OR logico)
il bit 5 = 1 per specificare che «1» logico è lo stato attivo da controllare
il bit 4 = 1 per indicare una parola di maschera viene di seguito (per annullare gli interrupt in sospeso).

Chiaramente potrebbe essere adoperata un'istruzione di Uscita di blocco ripetuta per abbreviare considerevolmente questi programmi.

INTERRUPT DEL PIO IN DAISY CHAIN

Ciascun PIO ha inoltre un'unica uscita per l'interrupt ed abilita i segnali destinati alla catena di priorità («daisy chain»). L'uscita $\overline{INT}$ è attiva bassa quando il PIO ha una richiesta d'interrupt. I segnali di abilitazione sono:

IEI (Ingresso d'Abilitazione all'Interrupt) — alto se nessun altro dispositivo di priorità più elevata è in quel momento servito da una routine di servizio dell'interrupt della CPU.

IEO (Uscita d'Abilitazione all'Interrupt) — alto se IEI è alto e se la CPU non sta gestando un interrupt da questo PIO.

SEGNALI DI DAISY CHAIN DEL PIO

IEI e IEO possono essere usati per costituire una catena di priorità (vedi Volume 1 di An Introduction to Microcomputers) nella quale i PIO e gli altri dispositivi più vicini alla CPU connessi in catena possono bloccare le richieste d'interrupt provenienti dai dispositivi più lontani dalla CPU. I pregi della catena di priorità sono:

1) Identifica ciascuna sorgente in maniera univoca.
2) Non richiede dell'altro hardware.
3) È facile da espandere o riaccomodare in hardware.

Gli svantaggi della catena di priorità sono i seguenti:

VANTAGGI E SVANTAGGI DEGLI INTERRUPT IN DAISY CHAIN

1) Può essere variata o mutata solamente in hardware.
2) Non provvede all'eventuale servizio degli interrupt a bassa priorità.
3) Richiede del tempo supplementare poichè i segnali si debbono propagare lungo la catena.

Lo Z80 attende automaticamente del tempo sufficiente affinché i segnali si propaghino lungo una catena composta al massimo di quattro dispositivi quando si opera nel Modo d'Interrupt 2. Si può aggiungere dell'hardware supplementare onde permettere delle catene più lunghe.

FUNZIONAMENTO DI UN DISPOSITIVO IN DAISY CHAIN

Notate che un dispositivo particolare nella catena funziona come segue:

1) Pone il suo vettore d'interrupt sul bus durante un ciclo di Riconoscimento dell'Interrupt solamente nel caso abbia in sospeso una richiesta d'interrupt e l'Ingresso di Abilitazione all'Interrupt sia a livello alto (il che indica che non si sta servendo in quell'istante alcun dispositivo di più elevata priorità). L'Uscita d'Abilitazione all'Interrupt viene inoltre posizionata a livello basso. All'interno di un dispositivo, gli interrupt della Porta A hanno la precedenza su quelli della Porta B.
2) Porta di conseguenza la sua Uscita d'Abilitazione all'Interrupt a livello alto (abilitando i dispositivi con priorità più bassa) solo nel caso venga eseguita un'istruzione RETI mentre il suo Ingresso d'Abilitazione all'Interrupt è a livello alto.

Così, un particolare dispositivo verrà servito solo quando esso ha la richiesta di priorità maggiore e bloccherà le richieste a priorità più bassa finché non è stata completata la sua routine di servizio. Un dispositivo con priorità più elevata può interrompere una routine di servizio di priorità inferiore senza alcuna difficoltà. Notate che un'istruzione RETI al termine della routine di priorità elevata non sarà riconosciuta dal dispositivo a priorità inferiore.

INTERRUPT DEL SIO

INTERRUPT DEL SIO

Il SIO può anche comportarsi come sorgente d'interrupt. Dovreste considerare le seguenti caratteristiche dei sistemi basati sull'interrupt del SIO:

1) L'interrupt del trasmettitore viene abilitato posizionando ad uno il bit 1 del Registro di Scrittura 1 di ciascun canale.
2) Il vettore d'interrupt viene influenzato dai bit 2, 3, e 4 del Registro di Scrittura 1 secondo le Tabelle 12-2 e 12-3.
3) Il vettore d'interrupt è contenuto nel Registro di Scrittura 2 del solo canale B. Esso può essere letto dal Registro di Lettura 2 dal solo Canale B.
4) Il bit D1 del registro di Lettura 0 del canale A è pari ad 1 se una qualsiasi condizione d'interrupt è presente nell'interno del SIO.

All'interno di un SIO, gli interrupt del Canale A hanno priorità su quelli del Canale B, gli interrupt del ricevitore hanno priorità su quelli del trasmettitore, e gli interrupt del trasmettitore hanno priorità su quelli esterni oppure su quelli di stato.

SISTEMI DI INTERRUPT A POLLING CON IL SIO

I dispositivi SIO possono essere usati in un sistema ad interrupt funzionante a scansione. La CPU deve controllare ciascun SIO esaminando l'attivazione del bit 1 del Registro di Lettura 9 nel canale A; cioè:

```
SUB     A            ;ACCEDI AL REGISTRO DI LETTURA 0
OUT     (SIOCRA),A
IN      A,(SIOCRA)   ;PRENDI LO STATO DEL SIO
BIT     1,A          ;C'È QUALCHE INTERRUPT IN SOSPESO?
JR      NZ,SERVE     ;SÌ, ATTIVA GLI INTERRUPT
```

Le caratteristiche importanti di un sistema a scansione per lo Z80 sono:

1) Il primo interrupt esaminato ha la priorità più elevata, dato che gli interrupt restanti non saranno esaminati se la prima è attiva. Il secondo interrupt ha la priorità successiva più elevata, e così via.
2) La routine di servizio deve annullare l'interrupt del SIO leggendo o scrivendo il registro dati appropriato anche se un trasferimento di dati è d'altra parte superfluo.

Tabella 12-2. Ulteriori Vettorizzazioni degli Interrupt del SIO (Il bit 2 del Registro di Scrittura 1 nel Canale B del SIO è uguale a 1)

Lo Stato Influenza il Vettore (D2) (Solo il Canale B)
Se questo bit è 1, il vettore di ritorno da un ciclo di riconoscimento degli interrupt varierà come segue:

	V3	V2	V1	
Ch B	0	0	0	Buffer di Trasmissione di Ch B Vuoto
	0	0	1	Cambiamento di Stato Esterno di Ch B
	0	1	0	Carattere Disponibile in Ricezione di Ch B
	0	1	1	Condizione Speciale in Ricezione di Ch B*
Ch A	1	0	0	Buffer di Trasmissione di Ch A Vuoto
	1	0	1	Cambiamento di Stato Esterno di Ch A
	1	1	0	Carattere Disponibile in Ricezione di Ch A
	1	1	1	Condizione Speciale in Ricezione di Ch A

*Condizioni Speciali in Ricezione →

Errore di Parità oppure
Errore di Overrun di Rx oppure
Errore di Framing/CRC oppure
Fine di Frame (SDLC)

Se questo bit è 0, ritorna il vettore fisso programmato nel registro del Vettore di Interrupt.

Tabella 12-3. Modi d'Interrupt del SIO (I bit 3 e 4 del Registro di Scrittura 1)

Rec Int Mode 0 (D3), Rec Int Mode 1 (D4)
Il Modo 0 di Interrupt in Ricezione e il Modo 1 di Interrupt in Ricezione specificano insieme le diverse condizioni di carattere disponibili:

Modo	D4 Rec Int Mode 1	D3 Rec Int Mode 0	
0	0	0	Interrupt del ricevitore disabilitati
1	0	1	Ricezione dell'interrupt solo su errore del primo carattere
2	1	0	Un interrupt su tutti i Caratteri in Ricezione con errore di Parità influenza il Vettore
3	1	1	Un interrupt su tutti i caratteri in Ricezione con errore di Parità non influenza il Vettore

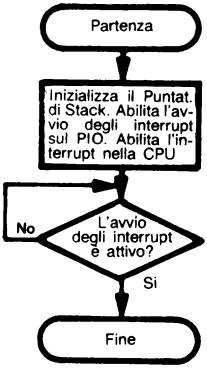
ESEMPI DI INTERRUPT

Un Interrupt di Avviamento

Scopo: Il computer attende che avvenga un interrupt da parte del PIO prima di dare inizio alle vere e proprie operazioni.

Molti sistemi restano inattivi finché l'operatore non li fa partire o finché non viene ricevuto un segnale DATA READY. Dopo l'attivazione del segnale $\overline{\text{RESET}}$, tali sistemi debbono inizializzare il Puntatore di Stack, abilitare l'interrupt di avviamento, ed eseguire un'istruzione HALT. Ricordate che $\overline{\text{RESET}}$ disabilita l'interrupt del processore e l'attivazione dell'alimentazione disabilita tutti gli interrupt del PIO. Nel diagramma di flusso, la decisione per verificare se l'avviamento è attivo viene presa dall'hardware (cioè dalla CPU che esamina internamente l'ingresso d'interrupt) invece che dal software.

Diagramma di Flusso:



Programma Sorgente:

Programma principale:

```
RESET   EQU     0
        ORG     RESET
        LD      SP,100H         ;METTI LO STACK ALLA FINE DELLA MEMORIA
        LD      A,01001111B     ;METTI IL PIO NEL MODO DI INGRESSO
        OUT     (PIOCRA),A
        LD      A,10000111B     ;ABILITA GLI INTERRUPT DEL PIO
        OUT     (PIOCRA),A
        EI                      ;ABILITA GLI INTERRUPT
        HALT                    ;E ASPETTA
```

Routine di servizio dell'interrupt:

```
        ORG     INTRP
        LD      SP,100H         ;RIINIZIALIZZA IL PUNTATORE DELLO STACK
        JP      START           ;FAI PARTIRE IL PROGRAMMA PRINCIPALE
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)	
Programma Principale:			
0000	31	LD	SP,100H
0001	00		
0002	01		
0003	3E	LD	A,01001111B
0004	4F		
0005	D3	OUT	(PIOCRA),A
0006	PIOCRA		
0007	3E	LD	A,10000111B
0008	87		
0009	D3	OUT	(PIOCRA),A
000A	PIOCRA		
000B	FB	EI	
000C	76	HALT	
Routine di Servizio dell'Interrupt:			
INTRP	31	LD	SP,100H
INTRP+1	00		
INTRP+2	01		
INTRP+3	C3	JP	START
INTRP+4			
INTRP+5	START		

Il programma principale deve inizializzare il Puntatore di Stack, dato che la risposta all'interrupt porta sempre alla memorizzazione del vecchio contenuto del Contatore del Programma nello Stack. A questo punto la routine di servizio inizializza semplicemente di nuovo il Puntatore di Stack prima che venga eseguita la reale routine di avviamento. Un metodo alternativo sarebbe quello di incrementare il Puntatore di Stack due volte prima di saltare alla routine di avviamento. Ricordate che lo Z80 si trova a funzionare nel Modo d'Interrupt 0. Qualsiasi altro modo richiederebbe l'esecuzione di un'istruzione IM.

INTERRUPT SU PARTICOLARI MICROCOMPUTER

L'esatta posizione della routine di servizio dell'interrupt varia da un microcomputer all'altro. Se il vostro microcomputer non ha alcun programma monitor, si può iniziare la routine di servizio dell'interrupt dovunque l'hardware esterno o la tabella del vettore diriga la CPU. Naturalmente, si dovrebbe collocare la routine in modo tale che non interferisca con degli indirizzi fissi o con altri programmi.

GESTIONE DEGLI INTERRUPT DA MONITOR

Se il vostro microcomputer possiede un programma monitor, questo occuperà sovente gli indirizzi di RESET e di servizio dell'interrupt. Esso quindi fornirà le routine di servizio o gli indirizzi di quelle routine. Una tipica inizializzazione della routine di monitor sarebbe:

```
MONIN:  PUSH    HL          ;SALVA IL VECCHIO CONTENUTO DEL REGISTRO
        LD      HL,USRINT   ;PRENDI L'INDIRIZZO DELL'UTILIZZATORE
                            ;   PER SERVIRLO
        JP      (HL)        ;SALTA ALL'INDIRIZZO DI SERVIZIO
                            ;   DELL'UTILIZZATORE
```

Dovete quindi collocare l'indirizzo della vostra routine di servizio nelle locazioni di memoria USRINT e USRINT+1, utilizzando il formato normale degli indirizzi dello Z80 con i bit meno significativi nell'indirizzo più basso. Ricordate che MONIN è un indirizzo nel programma monitor.

Potete includere il caricamento delle locazioni di memoria USRINT e USRINT+1 nel vostro programma principale: cioè:

```
LD      HL,INTRP      ;PRENDI L'INDIRIZZO DI PARTENZA
                      ;  DELLA ROUTINE DI SERVIZIO
LD      (USRINT),HL   ;IMMAGAZZINALO COME INDIRIZZO
                      ;  DELL'UTILIZZATORE
```

Queste istruzioni vengono prima dell'abilitazione degli interrupt.

In questo esempio, l'indirizzo di ritorno che lo Z80 memorizza nello Stack non è utile. Tuttavia, il programma principale deve ancora inizializzare il Puntatore di Stack cosicché vi è un posto definito dove porre quell'indirizzo. Potete fare a meno dell'istruzione LD SP se il monitor nel vostro microcomputer gestisce il Puntatore di Stack.

Il programma principale abilita soltanto l'interrupt dall'avviamento del PIO. Il PIO potrebbe, naturalmente, trovarsi in qualsiasi modo. L'interrupt viene abilitato posizionando ad uno il bit 7 di una parola di controllo dell'interrupt e scrivendo quella parola nella porta di controllo del PIO. L'interrupt del PIO viene abilitato prima che venga abilitato il sistema complessivo dell'interrupt mediante l'istruzione EI.

Ricordate che l'operazione di $\overline{\text{RESET}}$ e l'accettazione di un interrupt disabilitano automaticamente il sistema d'interrupt. Questo permette alla reale routine di avviamento di configurare tutti i PIO ed altre sorgenti d'interrupt senza essere interrotta.

Non è necessaria alcuna azione nella routine di servizio dell'interrupt, dato che l'interrupt viene automaticamente annullato come parte del ciclo di Riconoscimento dell'Interrupt che interessa un particolare PIO.

Le implementazioni delle istruzioni EI (Abilitazione degli Interrupt) e DI (Disabilitazione degli Interrupt) sono diverse nello Z80. DI ha effetto subito dopo la sua esecuzione, mentre EI ha effetto dopo l'esecuzione dell'istruzione seguente. Le motivazioni di questo comportamento sono trattate nel Capitolo 3 sotto la descrizione dell'istruzione EI.

Un Interrupt da Tastiera

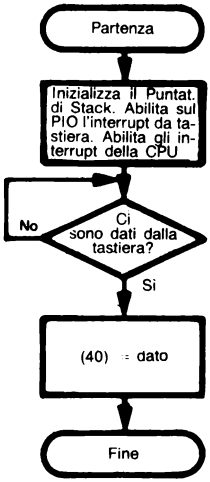
INTERRUPT DA TASTIERA

Scopo: Il computer attende un interrupt da una tastiera e pone il dato proveniente da quest'ultima nella locazione di memoria 0040.

Problema Campione:

Dato dalla tastiera = 06
Risultato: (0040) = 06

Diagramma di Flusso:



Programma Sorgente:

Programma Principale:

```
RESET   EQU     0
        ORG     RESET
        LD      SP,100H         ;METTI LO STACK ALLA FINE DELLA MEMORIA
        LD      A,01001111B     ;METTI IL PIO NEL MODO DI INGRESSO
        OUT     (PIOCRA),A
        LD      A,10000111B     ;ABILITA GLI INTERRUPT DEL PIO
        OUT     (PIOCRA),A
        EI                      ;ABILITA GLI INTERRUPT DELLA CPU
HERE:   JR      HERE            ;PROGRAMMA PRINCIPALE FITTIZIO
```

Routine di Servizio dell'Interrupt:

```
        ORG     INTRP
        EX      AF,AF'          ;SALVA L'ACCUMULATORE, I FLAG
        IN      A,(PIODRA)      ;PRENDI I DATI DELLA TASTIERA
        LD      (40H),A         ;SALVA I DATI DELLA TASTIERA
        EX      AF,AF'          ;RIMEMORIZZA L'ACCUMULATORE, I FLAG
        EI                      ;RIABILITA GLI INTERRUPT
        RETI
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)		Istruzione (Codice Mnemonico)	
Programma Principale:				
0000	31		LD	SP,100H
0001	00			
0002	01			
0003	3E		LD	A,01001111B
0004	4F			
0005	D3		OUT	(PIOCRA),A
0006	PIOCRA			
0007	3E		LD	A,10000111B
0008	87			
0009	D3		OUT	(PIOCRA),A
000A	PIOCRA			
000B	FB		EI	
000C	18	HERE:	JR	HERE
000D	FE			
Routine di Servizio dell'Interrupt:				
INTRP	08		EX	AF,AF'
INTRP+1	DB		IN	A,(PIODRA)
INTRP+2	PIODRA			
INTRP+3	32		LD	(40H),A
INTRP+4	40			
INTRP+5	00			
INTRP+6	08		EX	AF,AF'
INTRP+7	FB		EI	
INTRP+8	ED		RETI	
INTRP+9	4D			

L'istruzione JR HERE è un'istruzione ciclica interminabile (salto su sé stessa) che viene impiegata per rappresentare il programma principale. Dopo che vengono abilitati gli interrupt in un sistema funzionante, il programma principale si occupa dei compiti assegnatigli finché non avviene un interrupt e quindi riprende l'esecuzione dopo che la routine di servizio dell'interrupt viene completata.

CAMBIAMENTO DELL'INDIRIZZO DI RITORNO

L'istruzione RET al termine della routine di servizio trasferisce di nuovo il controllo all'istruzione JR. Se desiderate evitare tutto ciò, potete semplicemente incrementare il Contatore del Programma nello Stack, cioè,

```
EX      (SP),HL   ;PRENDI L'INDIRIZZO DI RITORNO
INC     HL        ;INCREMENTA DUE VOLTE L'INDIRIZZO DI RITORNO
INC     HL
EX      (SP),HL   ;RIMEMORIZZA L'INDIRIZZO ADATTATO PER LO STACK
```

L'istruzione RET cederà ora il controllo all'istruzione che segue JR. Notate l'uso di EX (SP), HL; questa istruzione scambia il contenuto della Coppia di Registri HL con il contenuto delle locazioni di memoria in cima allo Stack. Utilizzando quest'istruzione, si può adattare l'indirizzo senza influenzare il contenuto della Coppia di Registri HL.

Dato che lo Z80 non salva automaticamente i suoi registri, potete impiegarli per passare i parametri e i risultati tra il programma principale e la routine di servizio dell'interrupt. Così, si potrebbero lasciare i dati nell'Accumulatore invece che nella locazione di memoria 0040. Questo, tuttavia, è un metodo pericoloso che si dovrebbe evitare in tutti i sistemi tranne i più insignificanti.

Nella maggior parte di applicazioni, il processore utilizza i suoi registri durante l'esecuzione del normale programma; dato che le routine di servizio dell'interrupt cambiano casualmente il contenuto di quei registri, ciò provoca dello scompiglio. In generale, nessuna routine di servizio dell'interrupt dovrebbe mai alterare un qualsiasi registro a meno che i contenuti dei registri non siano stati salvati prima del loro mutamento ed essi verranno ripristinati al termine della routine.

Notate che si deve chiaramente riabilitare gli interrupt al termine della routine di servizio, dato che il processore disabilita il sistema d'interrupt quando accetta un interrupt. Quando si serve un interrupt del PIO si disattiva il segnale d'interrupt cosicchè lo stesso interrupt non viene servito un'altra volta.

SALVATAGGIO DI VALORI IN REGISTRI CON APICE

Se le routine di servizio dell'interrupt non vengono a loro volta interrotte (vale a dire esiste solo un livello degli interrupt), le istruzioni EX AF, AF' e EXX costituiscono un metodo conveniente per salvare e poi ripristinare i vecchi contenuti dei registri dell'utilizzatore. EXX scambia il contenuto di BC, DE e HL con il contenuto dei loro equivalenti alternativi. Le due istruzioni complessivamente richiedono due soli byte di memoria e otto cicli di clock. Comunque, questo metodo non può essere impiegato se ci sono altri livelli d'interrupt (dato che c'è solamente un unico gruppo di registri alternativi) oppure se i registri alternativi sono necessari nel programma principale o nella routine di servizio dell'interrupt.

Un metodo più generale per salvare e ripristinare i registri è quello di usare lo Stack. PUSH salva il contenuto di una coppia di registri e POP ne ripristina il contenuto. Tuttavia, PUSH richiede 11 cicli di clock e POP 10; pertanto questo sistema è più lento. Inoltre esso fa uso di ulteriori locazioni di memoria nello Stack. Il pregio di questo metodo è che esso può essere espanso illimitatamente (purché vi sia disponibilità di posto nello Stack) dato che le routine di servizio annidate non distruggeranno i dati salvati dalle precedenti routine.

RIEMPIMENTO DI UN BUFFER MEDIANTE INTERRUPT

Un metodo alternativo sarebbe che la routine d'interrupt mantenesse il controllo finché non si riceve una linea intera di testo (ad esempio una stringa di caratteri che finiscono con un ritorno carrello). Il programma principale sarebbe:

Programma Principale:

```
RESET   EQU     0
        ORG     RESET
        LD      SP,100H         ;METTI LO STACK ALLA FINE DELLA MEMORIA
        LD      A,01001111B     ;METTI IL PIO NEL MODO DI INGRESSO
        OUT     (PIOCRA),A
        LD      A,10000111B     ;ABILITA GLI INTERRUPT DEL PIO
        OUT     (PIOCRA),A
        LD      HL,70H          ;INIZIALIZZA IL PUNTATORE DEL BUFFER
        LD      (40H),HL        ;SALVA IL PUNTATORE DEL BUFFER
        EI                      ;ABILITA L'INTERRUPT DELLA CPU
HERE:   JR      HERE            ;PROGRAMMA PRINCIPALE FITTIZIO
```

Routine di Servizio degli Interrupt:

```
        ORG     INTRP
        EX      AF,AF'          ;SALVA A, I FLAG
        EXX                     ;SALVA GLI ALTRI REGISTRI
        LD      HL,(40H)        ;PRENDI IL PUNTATORE DEL BUFFER
        IN      A,(PIODRA)      ;PRENDI IL DATO DELLA TASTIERA
        LD      (HL),A          ;SALVA IL DATO NEL BUFFER
        CP      CR              ;IL DATO È UN RITORNO CARRELLO?
        JR      Z,ENDL          ;SÌ, FINE DELLA LINEA
        INC     HL              ;NO, INCREMENTA IL PUNTATORE DEL BUFFER
        LD      (40H),HL
        EXX                     ;RIMEMORIZZA GLI ALTRI REGISTRI
        EX      AF,AF'          ;RIMEMORIZZA A, I FLAG
        EI                      ;RIABILITA GLI INTERRUPT
        RETI
ENDL:   JP      LPROC           ;ESAMINA LA LINEA SENZA INTERRUPT
```

Quando il processore riceve un carattere di ritorno carrello, esso lascia disabilitato il sistema d'interrupt mentre si occupa della linea.

BUFFERING DOPPIO

Un modo alternativo sarebbe quello di riempire un altro buffer mentre si lavora con il primo; questo metodo è noto come doppio buffering.

La routine di elaborazione della linea viene iniziata all'indirizzo LPROC con gli interrupt disabilitati; col contenuto dei vecchi registri alternativi, e con l'indirizzo di ritorno originale in cima allo Stack.

In una reale applicazione, la CPU potrebbe compiere altre attività tra gli interrupt. Potrebbe, per esempio, redarre editing, spostare, o trasmettere una linea da un buffer mentre l'interrupt sta riempendo un altro buffer.

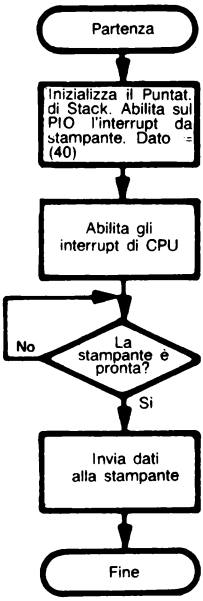
Un Interrupt da Stampante

Scopo: Il computer attende un interrupt da una stampante e invia il dato dalla locazione di memoria 0040 alla stampante.

Problema Campione:

(0040) = 51H
Risultato: La stampante riceve 51H (Q in codice ASCII) quando è pronta.

Diagramma di Flusso:



Programma Sorgente:

Programma Principale:

```
RESET   EQU     0
        ORG     RESET
        LD      SP,100H         ;METTI LO STACK ALLA FINE DELLA MEMORIA
        LD      A,00001111B     ;METTI IL PIO NEL MODO DI USCITA
        OUT     (PIOCRA),A
        LD      A,10000111B     ;ABILITA GLI INTERRUPT DEL PIO
        OUT     (PIOCRA),A
        EI                      ;ABILITA GLI INTERRUPT DELLA CPU
HERE:   JR      HERE            ;PROGRAMMA PRINCIPALE FITTIZIO
```

Routine di Servizio degli Interrupt:

```
        ORG     INTRP
        EX      AF,AF'          ;SALVA L'ACCUMULATORE, I FLAG
        LD      A,(40H)         ;PRENDI IL DATO
        OUT     (PIODRA),A      ;INVIA IL DATO ALLA STAMPANTE
        EX      AF,AF'          ;RIMEMORIZZA L'ACCUMULATORE, I FLAG
        EI                      ;RIABILITA GLI INTERRUPT
        RETI
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)		
Programma Principale:				
0000	31		LD	SP,100H
0001	00			
0002	01			
0003	3E		LD	A,00001111B
0004	0F			
0005	D3		OUT	(PIOCRA),A
0006	PIOCRA			
0007	3E		LD	A,10000111B
0008	87			
0009	D3		OUT	(PIOCRA),A
000A	PIOCRA			
000B	FB		EI	
000C	18	HERE:	JR	HERE
000D	FE			
Routine di Servizio dell'Interrupt:				
INTRP	08		EX	AF,AF'
INTRP+1	3A		LD	A,(40H)
INTRP+2	40			
INTRP+3	00			
INTRP+4	D3		OUT	(PIODRA),A
INTRP+5	PIODRA			
INTRP+6	08		EX	AF,AF'
INTRP+7	FB		EI	
INTRP+8	ED		RETI	
INTRP+9	4D			

SVUOTAMENTO DI UN BUFFER CON INTERRUPT

In questo caso, come con la tastiera, ci potrebbe essere la possibilità che la stampante continui ad interrompere finché non viene trasferita un'intera linea di testo. Il programma principale e la routine di servizio sarebbero:

Programma Principale:

```
RESET   EQU     0
        ORG     RESET
        LD      SP,100H         ;METTI LO STACK ALLA FINE DELLA MEMORIA
        LD      A,00001111B     ;METTI IL PIO NEL MODO DI USCITA
        OUT     (PIOCRA),A
        LD      A,10000111B     ;ABILITA GLI INTERRUPT DEL PIO
        OUT     (PIOCRA),A
        LD      HL,70H          ;INIZIALIZZA IL PUNTATORE DEL BUFFER
        LD      (40H),HL        ;SALVA IL PUNTATORE DEL BUFFER
        EI                      ;ABILITA GLI INTERRUPT DELLA CPU
HERE:   JR      HERE            ;PROGRAMMA PRINCIPALE FITTIZIO
```

Routine di Servizio degli Interrupt:

```
        ORG     INTRP
        EX      AF,AF'          ;SALVA A, I FLAG
        EXX                     ;SALVA GLI ALTRI REGISTRI
        LD      HL,(40H)        ;PRENDI IL PUNTATORE DEL BUFFER
        LD      A,(HL)          ;PRENDI UN BYTE DEL DATO DAL BUFFER
        OUT     (PIODRA),A      ;INVIA IL DATO ALLA STAMPANTE
        CP      CR              ;IL DATO È UN RITORNO CARRELLO?
        JR      ENDL            ;SÌ, FINE DELLA LINEA
        INC     HL              ;NO, INCREMENTA IL PUNTATORE DEL BUFFER
        LD      (40H),HL
        EXX                     ;RIMEMORIZZA GLI ALTRI REGISTRI
        EX      AF,AF'          ;RIMEMORIZZA A, I FLAG
        EI                      ;RIABILITA GLI INTERRUPT
        RETI
ENDL:   JP      LCOMP           ;GESTISCI LA LINEA COMPLETATA
```

Inoltre, si potrebbe usare la tecnica del doppio buffering per fare in modo che avvengano nello stesso tempo le operazioni dell'I/O e di elaborazione senza mai fermare la CPU.

Un Interrupt da un Clock in Tempo Reale

Scopo: Il computer resta in attesa di un interrupt da un clock in tempo reale.

CLOCK IN TEMPO REALE

Un clock in tempo reale fornisce una serie regolare di impulsi. L'intervallo tra gli impulsi può essere usato come riferimento di tempo. Possono essere contati gli interrupt da clock in tempo reale per fornire qualsiasi multiplo dell'intervallo di tempo base. Si può produrre un clock in tempo reale dividendo il clock della CPU, mediante l'uso di un temporizzatore separato o di un temporizzatore programmabile come il CTC per i microcomputer basati sullo Z80, o mediante delle sorgenti esterne quali la frequenza della linea CA.

FREQUENZA DEL CLOCK IN TEMPO REALE

Notate le specifiche che sono implicate nel determinare la frequenza del clock in tempo reale. Una frequenza elevata (diciamo 10KHz) permette la creazione di un'ampia variazione di intervalli di tempo con alta precisione. D'altra parte, il lavoro di supervisione che consegue al conteggio delle istruzioni da un clock in tempo reale può risultare considerevole, e i conteggi supereranno rapidamente la capacità di un solo registro o cella di memoria ad 8 bit. La scelta della frequenza dipende dalla precisione e dai requisiti di temporizzazione della vostra applicazione. Il clock può, naturalmente, consistere in parte di hardware; un contatore può contare impulsi ad alta frequenza e interrompere il processore soltanto saltuariamente. Un programma avrà il compito di leggere il contatore per misurare il tempo con elevata precisione.

SINCRONIZZAZIONE CON CLOCK IN TEMPO REALE

Uno dei problemi è l'operazione di sincronizzazione con il clock in tempo reale. Logicamente, si influenzerà la precisione dell'intervallo di tempo se la CPU dà inizio alla misura in modo casuale durante un periodo di clock, anziché esattamente all'inizio dello stesso periodo. Alcuni metodi per sincronizzare le operazioni sono:

1) Avviare la CPU ed il clock simultaneamente. Il RESET o un interrupt d'avviamento possono inizializzare sia il clock che la CPU.
2) Dare l'opportunità alla CPU di avviare e di bloccare il clock sotto il controllo del programma.
3) Utilizzare un clock ad alta frequenza in modo tale che un errore inferiore a un periodo di clock risulterà trascurabile.
4) Allineare il clock (in attesa di un fronte attivo o di un interrupt) prima di iniziare la misura.

PRIORITÀ DI CLOCK IN TEMPO REALE

Un interrupt da clock in tempo reale dovrebbe avere una priorità molto elevata, poiché la precisione degli intervalli di temporizzazione verrà influenzata da un qualsiasi ritardo nel servizio dell'interrupt. La regola consueta è quella di rendere il clock in tempo reale l'interrupt a priorità più elevata a parte la mancanza d'alimentazione. La routine di servizio dell'interrupt viene mantenuta generalmente il più possibile breve in modo che essa non interferisca con le altre attività della cPU.

a) Attesa del Clock in Tempo Reale

Programma Sorgente:

Programma Principale:

```
RESET   EQU     0
        ORG     RESET
        LD      SP,100H         ;METTI LO STACK ALLA FINE DELLA MEMORIA
        LD      A,01001111B     ;METTI IL PIO NEL MODO DI INGRESSO
        OUT     (PIOCRA),A
        LD      A,10000111B     ;ABILITA GLI INTERRUPT DEL PIO
        OUT     (PIOCRA),A
        EI                      ;ABILITA GLI INTERRUPT DELLA CPU
HERE:   JR      HERE            ;PROGRAMMA PRINCIPALE FITTIZIO
```

Routine di Servizio degli Interrupt:

```
        ORG     INTRP
        HALT                    ;FINE DEGLI INTERRUPT DA CLOCK
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)		
Programma Principale:				
0000	31		LD	SP,100H
0001	00			
0002	01			
0003	3E		LD	A,01001111B
0004	4F			
0005	D3		OUT	(PIOCRA),A
0006	PIOCRA			
0007	3E		LD	A,10000111B
0008	87			
0009	D3		OUT	(PIOCRA),A
000A	PIOCRA			
000B	FB		EI	
000C	18	HERE:	JR	HERE
000D	FE			
Routine di Servizio dell'Interrupt:				
INTRP	76			HALT

La routine di servizio non deve far nulla, poiché quando ci serve l'interrupt del PIO essa si annulla automaticamente e non vi sono dati da trasmettere o ricevere.

L'interrupt da un clock in tempo reale capita sempre su un fronte di salita se si utilizza un segnale PIO STROBE come ingresso di clock.

b) Attesa di 10 Interrupt dal Clock in Tempo Reale

Programma Sorgente:

Programma Principale:

```
RESET   EQU     0
        ORG     RESET
        LD      SP,100H         ;METTI LO STACK ALLA FINE DELLA MEMORIA
        LD      A,01001111B     ;METTI IL PIO NEL MODO DI INGRESSO
        OUT     (PIOCRA),A
        LD      A,10000111B     ;ABILITA GLI INTERRUPT DEL PIO
        OUT     (PIOCRA),A
        LD      HL,40H          ;CONTATORE DI CLOCK = ZERO
        LD      (HL),0
        LD      A,10            ;NUMERO DI CONTEGGI = 10
        EI                      ;ABILITA GLI INTERRUPT DELLA CPU
WTTEN:  CP      (HL)            ;SONO PASSATI DIECI CONTEGGI?
        JR      NZ,WTTEN        ;NO, ASPETTA
        HALT                    ;SÌ
```

Routine di Servizio degli Interrupt:

```
        ORG     INTRP
        EXX                     ;SALVA I REGISTRI DELL'UTILIZZATORE
        EX      AF,AF'          ;SALVA A, I FLAG
        LD      HL,40H          ;INCREMENTA IL CONTATORE DI CLOCK
        INC     (HL)
        EX      AF,AF'          ;RIMEMORIZZA A, I FLAG
        EXX                     ;RIMEMORIZZA I REGISTRI DELL'UTILIZZATORE
        EI                      ;RIABILITA GLI INTERRUPT
        RETI
```

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)	
Programma Principale:			
0000	31	LD	SP,100H
0001	00		
0002	01		
0003	3E	LD	A,01001111B
0004	4F		
0005	D3	OUT	(PIOCRA),A
0006	PIOCRA		
0007	3E	LD	A,10000111B
0008	87		
0009	D3	OUT	(PIOCRA),A
000A	PIOCRA		
000B	21	LD	HL,40H
000C	40		
000D	00		
000E	36	LD	(HL),0
000F	00		
0010	3E	LD	A,10
0011	0A		
0012	FB	EI	
0013	BE	WTTEN: CP	(HL)
0014	20	JR	NZ,WTTEN
0015	FD		
0016	76	HALT	
Routine di Servizio dell'Interrupt:			
INTRP	D9	EXX	
INTRP+1	08	EX	AF,AF'
INTRP+2	21	LD	HL,40H
INTRP+3	40		
INTRP+4	00		
INTRP+5	34	INC	(HL)
INTRP+6	08	EX	AF,AF'
INTRP+7	D9	EXX	
INTRP+8	FB	EI	
INTRP+9	ED	RETI	
INTRP+10	4D		

Un metodo diverso impiega lo Stack per conservare e poi ripristinare il valore dei registri. Per salvare H, L ed i flag è necessario ciò che segue:

```
PUSH    HL      ;SALVA I REGISTRI H ED L
PUSH    AF      ;SALVA L'ACCUMULATORE ED I FLAG
```

Per ripristinarli bisogna eseguire la sequenza:

```
POP     AF      ;RIPRISTINA L'ACCUMULATORE ED I FLAG
POP     HL      ;RIPRISTINA I REGISTRI H ED L
```

Notate che, se viene impiegato lo Stack, i registri debbono essere ripristinati nell'ordine opposto a quello nel quale erano stati salvati. Chiaramente l'ordine nel quale sono eseguite EXX e EX AF,AF' non ha importanza.

Questa routine di servizio dell'interrupt aggiorna solamente il contatore contenuto nella locazione di memoria 0040. Essa è trasparente rispetto al programma principale.

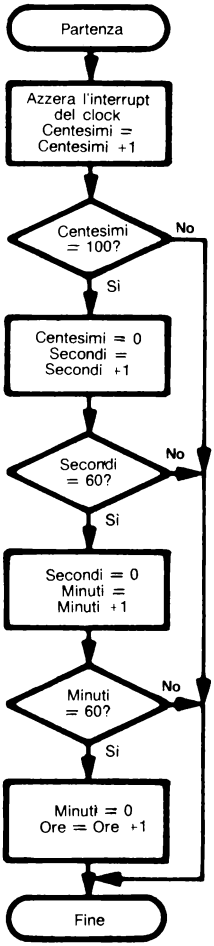
CONSERVAZIONE DEL TEMPO REALE

Una routine d'interrupt da un clock in tempo reale più verosimile potrebbe mantenere il tempo reale in diverse locazioni di memoria. Per esempio, la routine che segue impiega gli indirizzi da 0040 a 0043 in questo modo:

0040 - centesimi di secondo
0041 - secondi
0042 - minuti
0043 - ore

Supponiamo che la routine venga innescata da un clock a 100 Hz.

Diagramma di Flusso:



Programma Sorgente:

```
        ORG     INTRP
        PUSH    AF          ;SALVA I REGISTRI
        PUSH    HL
        LD      HL,40H      ;AGGIORNA I CENTESIMI DI SECONDO
        INC     (HL)
        LD      A,100
        CP      (HL)        ;C'È RIPORTO PER I SECONDI?
        JR      NZ,DONE     ;NO, VAI A DONE
        LD      (HL),0      ;SÌ, CENTESIMI = 0
        INC     HL          ;AGGIORNA I SECONDI
        INC     (HL)
        LD      A,60
        CP      (HL)        ;C'È RIPORTO PER I MINUTI?
        JR      NZ,DONE     ;NO, VAI A DONE
        LD      (HL),0      ;SÌ, SECONDI = 0
        INC     HL          ;AGGIORNA I MINUTI
        INC     (HL)
        CP      (HL)        ;C'È RIPORTO PER LE ORE?
        JR      NZ,DONE     ;NO, VAI A DONE
        LD      (HL),0      ;SÌ, MINUTI = 0
        INC     HL          ;AGGIORNA LE ORE
        INC     (HL)
        LD      A,24        ;IL GIORNO È COMPLETATO?
        JR      NZ,DONE     ;NO, VAI A DONE
        LD      (HL),0      ;SÌ, ORE = 0
DONE:   POP     HL          ;RIMEMORIZZA I REGISTRI
        POP     AF
        EI                  ;RIABILITA GLI INTERRUPT
        RETI
```

Ora si può produrre un'attesa di 300 msec nel programma principale con la routine:

```
        LD      HL,40H      ;PRENDI IL TEMPO PRESENTE
                            ;   (CENTINAIA DI SECONDI)
        LD      A,(HL)
        ADD     A,30        ;IL TEMPO DESIDERATO È 30
                            ;   CONTEGGI IN RITARDO
        CP      100         ;MOD 100
        JR      C,WT30
        SUB     100
WT30:   CP      (HL)        ;SI È ARRIVATI AL TEMPO DESIDERATO?
        JR      NZ,WT30     ;NO, ASPETTA
```

Notate in questo programma la differenza tra INC HL e INC (HL). INC HL aggiunge 1 al contenuto a 16 bit della Coppia di Registri HL, mentre INC (HL) aggiunge 1 al contenuto ad 8 bit della locazione di memoria indirizzata da HL.

Naturalmente, il programma potrebbe eseguire altri compiti e controllare il tempo trascorso soltanto di tanto in tanto. Come produrreste un ritardo di sette secondi? Di tre minuti?

Talvolta potete desiderare di conservare il tempo sotto forma o di cifre BCD o di caratteri ASCII. Come ristrutturereste il precedente programma per trattare questi metodi alternativi?

Si può disabilitare l'interrupt da parte del clock (o qualsiasi altro interrupt) quando esso non è più necessario con uno qualsiasi dei modi seguenti.

DISABILITAZIONE DEGLI INTERRUPT

1) Eseguendo un'istruzione DI nel programma principale. Ciò disabilita l'intero sistema d'interrupt.
2) Azzerando il bit 7 della parola di controllo dell'interrupt durante la routine di servizio o durante il programma principale. Ciò disabilita solamente l'interrupt da un'unica porta di un unico PIO.
3) Non riabilitando l'interrupt durante la routine di servizio.

Ricordate che la CPU disabilita automaticamente gli interrupt quando ne accetta uno. Così, il sistema d'interrupt viene disabilitato a meno che la routine di servizio non lo riabiliti esplicitamente. Notate, tuttavia, che si deve stare attenti alla riabilitazione degli interrupt, dato che il programma principale sarebbe completamente all'oscuro del fatto che gli interrupt non sono stati più permessi. In generale, tutte le routine di servizio dell'interrupt dovrebbero riabilitare gli interrupt prima di ritornare alla gestione del programma principale; qualsiasi altra metodologia usata significa che le routine di servizio non sono trasparenti rispetto al programma principale.

Un Interrupt da Telescrivente

Scopo: Il computer attende la ricezione di un dato da una telescrivente e memorizza il dato stesso nella locazione di memoria 0040.

a) Utilizzazione di un SIO

(caratteri di 7 bit con parità di tipo dispari e 2 bit di stop).

ROUTINE DI INTERRUPT DEL SIO

Programma Sorgente:

Programma Principale:

```
RESET   EQU     0
        LD      A,4             ;ACCEDI AL REGISTRO DI SCRITTURA 4
        OUT     (SIOCRA),A
        LD      A,01000001B     ;MODO DI CLOCK X 16, PARITÀ
        OUT     (SIOCRA),A
        LD      A,3
        OUT     (SIOCRA),A
        LD      A,01000001B     ;CARATTERI A 7 BIT, ABILITA IL RICEVITORE
        OUT     (SIOCRA),A
        LD      A,1             ;ACCEDI AL REGISTRO DI SCRITTURA 1
        OUT     (SIOCRA),A
        LD      A,00011000B     ;ABILITA L'INTERRUPT DEL RICEVITORE
                                ;  SU TUTTI I CARATTERI
        OUT     (SIOCRA),A
        EI                      ;ABILITA GLI INTERRUPT DELLA CPU
HERE:   JR      HERE            ;PROGRAMMA PRINCIPALE FITTIZIO
```

Routine di Servizio degli Interrupt:

```
        ORG     INTRP
        PUSH    AF              ;SALVA L'ACCUMULATORE, I FLAG
        IN      A,(SIODRA)      ;LEGGI IL CARATTERE DAL SIO
        LD      (40H),A         ;SALVA IL CARATTERE IN MEMORIA
        POP     AF              ;RIMEMORIZZA L'ACCUMULATORE, I FLAG
        EI                      ;RIABILITA GLI INTERRUPT
        RETI
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)		Istruzione (Codice Mnemonico)	
Programma Principale:				
0000	3E		LD	A,4
0001	04			
0002	D3		OUT	(SIOCRA),A
0003	SIOCRA			
0004	3E		LD	A,01000001B
0005	41			
0006	D3		OUT	(SIOCRA),A
0007	SIOCRA			
0008	3E		LD	A,3
0009	03			
000A	D3		OUT	(SIOCRA),A
000B	SIOCRA			
000C	3E		LD	A,01000001B
000D	41			
000E	D3		OUT	(SIOCRA),A
000F	SIOCRA			
0010	3E		LD	A,1
0011	01			
0012	D3		OUT	(SIOCRA),A
0013	SIOCRA			
0014	3E		LD	A,00011000B
0015	18			
0016	D3		OUT	(SIOCRA),A
0017	SIOCRA			
0018	FB		EI	
0019	18	HERE:	JR	HERE
001A	FE			
Routine di Servizio dell'Interrupt:				
INTRP	F5		PUSH	AF
INTRP+1	DB		IN	A,(SIODRA)
INTRP+2	SIODRA			
INTRP+3	32		LD	(40H),A
INTRP+4	40			
INTRP+5	00			
INTRP+6	F1		POP	AF
INTRP+7	FB		EI	
INTRP+8	ED		RETI	
INTRP+9	4D			

Questa routine di servizio presuppone che solamente l'interrupt del ricevitore da un canale del SIO è stata abilitata. Altrimenti, o si renderà necessaria un'ulteriore vettorizzazione variando i bit di controllo D2, D3, e D4 del Registro di Scrittura 0 (vedi la precedente trattazione degli interrupt in questo capitolo) o la routine dovrà esaminare i bit di stato nel Registro di Lettura 0. I bit di stato chiave sono:

BIT 0 - Disponibilità del Carattere nel Ricevitore - 1 quando almeno un carattere è disponibile nei buffer di ricezione.

BIT 1 - Interrupt in sospeso (solo Canale A) - 1 se un qualsiasi interrupt è in sospeso in tutto il SIO.

BIT 2 - Buffer del Trasmettitore Vuoto - 1 se il «buffer» di Trasmissione è vuoto.

Evidentemente, sarebbe di gran lunga più breve e più semplice configurare il SIO mediante l'uso di una tabella (in ROM) e dell'istruzione I/O di Blocco ripetuta, vale a dire:

```
LD      B,6         ;NUMERO DI BYTE DELLA CONFIGURAZIONE
LD      C,SIOCRA    ;PORTA DI CONTROLLO DEL SIO
LD      HL,SIOTBL   ;PARTENZA DELLA TABELLA DELLE
                    ;  CONFIGURAZIONI DEL SIO
OTIR                ;CONFIGURA IL SIO
```

Questo metodo richiede 9 byte di memoria per il programma e 6 byte per la tabella, da confrontare con i 23 byte usati nell'esempio per configurare il SIO.

Il programma organizza i registri nel modo seguente:

REGISTRO DI SCRITTURA 4

Bit 7 = 0, bit 6 = 1 per il modo di clock x 16
Bit 1 = 0 per selezionare la parità di tipo dispari
Bit 0 = 1 per abilitare la generazione della parità

REGISTRO DI SCRITTURA 3

Bit 7 = 0, bit 6 = 1 per selezionare caratteri di 7 bit
Bit 0 = 1 per abilitare il ricevitore

REGISTRO DI SCRITTURA 1

Bit 4 = 1, bit 3 = 1 per produrre un interrupt su tutti i caratteri ricevuti senza che gli errori di parità influenzino il vettore.

La CPU azzera il bit di Disponibilità del Carattere Ricevuto leggendo un carattere dal registro Dati del SIO. Il bit d'Interrupt in Sospeso viene azzerato automaticamente quando è servito l'interrupt.

b) Utilizzazione di un PIO

(Dati ricevuti connessi al bit 7 dei dati relativi alla Porta A del PIO).

Programma Sorgente:

Programma Principale:

```
        LD      A,11001111B   ;RENDI LA PORTA A UN CONTROLLO
        OUT     (PIOCRA),A
        LD      A,10000000B   ;RENDI IL BIT 7 UN INGRESSO, GLI ALTRI USCITE
        OUT     (PIOCRA),A
        LD      A,10010111B   ;ABILITA L'INTERRUPT SUL BIT DI PARTENZA (0)
        OUT     (PIOCRA),A
        LD      A,01111111B   ;MASCHERA TUTTI GLI ALTRI BIT
        OUT     (PIOCRA),A
        EI                    ;ABILITA GLI INTERRUPT DELLA CPU
HERE:   JR      HERE          ;PROGRAMMA PRINCIPALE FITTIZIO
```

Routine di Servizio degli Interrupt:

```
        ORG     INTRP
        PUSH    AF            ;SALVA L'ACCUMULATORE, I FLAG
        LD      A,00000111B   ;DISABILITA L'INTERRUPT DEL BIT DI PARTENZA
        OUT     (PIOCRA),A
        CALL    TTYRCV        ;VAI A PRENDERE IL DATO DA TTY
        LD      A,10000111B   ;ABILITA L'INTERRUPT DEL BIT DI PARTENZA
        OUT     (PIOCRA),A
        POP     AF            ;RIMEMORIZZA L'ACCUMULATORE, I FLAG
        EI                    ;RIABILITA GLI INTERRUPT
        RETI
```

Programma Oggetto:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)		Istruzione (Codice Mnemonico)	
Programma Principale:				
0000	3E		LD	A,11001111B
0001	CF			
0002	D3		OUT	(PIOCRA),A
0003	PIOCRA			
0004	3E		LD	A,10000000B
0005	80			
0006	D3		OUT	(PIOCRA),A
0007	PIOCRA			
0008	3E		LD	A,10010111B
0009	97			
000A	D3		OUT	(PIOCRA),A
000B	PIOCRA			
000C	3E		LD	A,01111111B
000D	7F			
000E	D3		OUT	(PIOCRA),A
000F	PIOCRA			
0010	FB		EI	
0011	18	HERE:	JR	HERE
0012	FE			
Routine di Servizio dell'Interrupt:				
INTRP	F5		PUSH	AF
INTRP+1	3E		LD	A,00000111B
INTRP+2	07			
INTRP+3	D3		OUT	(PIOCRA),A
INTRP+4	PIOCRA			
INTRP+5	CD		CALL	TTYRCV
INTRP+6	TTYRCV			
INTRP+7				
INTRP+8	3E		LD	A,10000111B
INTRP+9	87			
INTRP+10	D3		OUT	(PIOCRA),A
INTRP+11	PIOCRA			
INTRP+12	F1		POP	AF
INTRP+13	FB		EI	
INTRP+14	ED		RETI	
INTRP+15	4D			

Questi programmi suppongono che il monitor inizializzi il Puntatore di Stack. Altrimenti, questo dovrà essere caricato nel programma principale.

La subroutine TTYRCV è la routine di servizio della TTY in ricezione già considerata nel precedente capitolo.

Il fronte impiegato per provocare l'interrupt è qui molto importante. Un interrupt deve avvenire quando la linea dei dati varia dal normale «MARK» o stato '1' allo «SPACE» o stato '0', dato che questa transizione identifica l'inizio della trasmissione.

La routine di servizio deve disabilitare l'interrupt del PIO, poiché altrimenti ciascuna transizione da '1' a '0' nel carattere porterà ad un interrupt. Naturalmente, si deve riabilitare l'interrupt del PIO dopo che è stato letto il carattere completo.

Notate l'uso del PIO nel modo di controllo:

1) Il PIO viene posizionato nel modo di controllo imponendo il Modo 3.
2) La successiva parola di controllo definisce quali linee dei dati debbono essere considerate come ingressi ('1') e quali come uscite ('0').
3) La parola di controllo dell'interrupt ha, oltre alla solita abilitazione nel bit 7,
   bit 6 = 0 per eseguire un OR logico delle linee dei dati controllate in attesa di un interrupt (non usato in questo caso, dato che viene esaminata solo una linea)
   bit 5 = 0 per definire che la polarità attiva delle linee dei dati è il valore basso (per il bit di start in questo caso).
   bit 4 = 1 per indicare che segue una parola di maschera.

4) La parola di controllo subito seguente contiene le maschere dell'interrupt. Soltanto le linee della porta che hanno un bit di maschera a zero verranno controllate per la generazione di un interrupt.

Ciò che ne risulta è che per dare origine a un interrupt il bit 7 deve essere zero o variare da uno a zero. Notate che gli ulteriori interrupt avvengono solo in concomitanza di una variazione nello stato dell'equazione logica. Anche in questo caso, il PIO potrebbe essere configurato utilizzando una tabella e l'istruzione di uscita di blocco ripetuta.

ROUTINE DI SERVIZIO PIÙ GENERALI

Le routine di servizio più generali che fanno parte di un sistema completo pilotato ad interrupt devono eseguire i seguenti compiti:

COMPITI PER ROUTINE DI SERVIZIO GENERALI

1) Salvataggio di tutti i registri che sono impiegati nella routine di servizio all'interno dello Stack in modo che il programma interrotto possa essere ripreso in modo corretto.

Ricordate che l'istruzione «Push» dello Z80 trasferisce una coppia di registri (oppure un registro indice) nello Stack. PUSH AF (F è il registro di Flag) trasferisce l'Accumulatore ed i flag nello Stack.

La routine per salvare tutti i registri nello Stack sarà la seguente:

```
PUSH    AF      ;SALVA L'ACCUMULATORE, I FLAG
PUSH    BC      ;SALVA I REGISTRI B, C
PUSH    DE      ;SALVA I REGISTRI D, E
PUSH    HL      ;SALVA I REGISTRI H, L
PUSH    IX      ;SALVA IL REGISTRO INDICE IX
PUSH    IY      ;SALVA IL REGISTRO INDICE IY
EX      AF,AF'
EXX
PUSH    AF      ;SALVA L'ACCUMULATORE CON APICE, I FLAG
PUSH    BC      ;SALVA I REGISTRI CON APICE B, C
PUSH    DE      ;SALVA I REGISTRI CON APICE D, E
PUSH    HL      ;SALVA I REGISTRI CON APICE H, L
```

Naturalmente, devono essere salvati solamente quei registri che vengono poi usati dalla routine di servizio dell'interrupt.

2) Ripristino di tutti i registri dallo Stack dopo aver completato la routine di servizio dell'interrupt. Ricordate che i registri debbono essere ripristinati nell'ordine opposto a quello nel quale erano stati salvati.
3) Abilitazione e disabilitazione degli interrupt in modo appropriato. Ricordate che la CPU disabilita automaticamente i suoi interrupt nell'istante stesso in cui ne accetta una.

Le routine di servizio dovrebbero essere trasparenti per quanto riguarda il programma interrotto (cioè esse non dovrebbero avere nessun effetto secondario).

Qualsiasi siano le subroutine standard impiegate da una routine di servizio dell'interrupt esse devono essere rientranti. Se qualche subroutine non lo è, la routine di servizio dell'interrupt deve avere versioni distinte per ogni uso.[5]

PROBLEMI

1) Un Interrupt di Test

Scopo: Il computer resta in attesa che avvenga un interrupt da parte del PIO, quindi esegue l'istruzione di un ciclo senza fine:

```
HERE:   JR      HERE
```

finché non capita il successivo interrupt.

2) Un Interrupt da Tastiera

Scopo: Il computer attende un invio di 4 cifre da una tastiera e pone le cifre nelle locazioni di memoria da 0040 fino a 0043 (la prima che viene ricevuta in 0040). Ciascun invio di cifra provoca un interrupt. Il quarto invio dovrebbe inoltre portare come conseguenza la disabilitazione dell'interrupt dalla tastiera.

Problema Campione:

```
Keyboard data  =  04, 06, 01, 07
Risultato: (0040)  =  04
           (0041)  =  06
           (0042)  =  01
           (0043)  =  07
```

3) Un Interrupt da Stampante

Scopo: Il computer spedisce quattro caratteri dalla locazione di memoria 0040 fino a 0043 (a partire da 0040) verso la stampante. Ciascun carattere viene richiesto tramite un interrupt. Il quarto trasferimento disabilita inoltre l'interrupt dalla stampante.

4) Un Interrupt da Clock in Tempo Reale

Scopo: Il computer azzera inizialmente la cella di memoria 0040 e poi complementa questa stessa cella ogni volta che avviene l'interrupt da un clock in tempo reale.

Come trasferire il programma per fare in modo che esso complementi la cella 0040 dopo ogni dieci interrupt? Come trasformereste il programma in modo che esso lasci la cella 0040 a zero per dieci periodi di clock, a FF_{16} per cinque periodi di clock, e così via di continuo? Avete la possibilità di impiegare un visualizzatore al posto della cella di memoria 0040 cosicché il fenomeno sarà più semplice da seguire.

5) Un Interrupt da Telescrivente

Scopo: Il computer riceve dei dati TTY da un SIO che interrompe e memorizza i caratteri in un «buffer» che inizia dalla cella di memoria 0040. Il processo continua finché il computer riceve un carattere di ritorno carrello (OD_{16}).

Supponete che i caratteri siano in codice ASCII a 7 bit con parità di tipo dispari. Come trasformereste il vostro programma in modo da utilizzare un PIO? Supponete che sia disponibile la subroutine TTYRCV, come nell'esempio. Includere il ritorno carrello come carattere finale nel buffer.

BIBLIOGRAFIA

1. Si può riesaminare la trattazione degli interrupt nel Volume 1 di An Introduction To Microcomputer.

2. Per una trattazione di progetti con interrupt, vedere R. L. Baldridge, "Interrupts Add Power, Complexity to Microcomputer System Design," EDN, 5 Agosto 1977, pp. 67-73.

3. Vedi Volume 2 di An Introduction to Microcomputers.

4. Vedi An Introduction to Microcomputers, Volume 2 and 8080A/8085 Assembly Language Programming.

5. Per un'ulteriore trattazione e per alcuni esempi reali di progettazione di sistemi ad interrupt basati sullo Z80, vedere da pagina 5-24 a 5-37 di Z80 Programmazione per il Progetto Logico ed i seguenti:
   Baldridge, R. L., "Interrupts Add Power, Complexity to Microcomputer System Design," EDN, 5 Agosto 1977, pp. 67-73.
   Pond, R. M., "Let Microprocessors Communicate," Electronic Design, 8 Novembre, 1977, pp. 88-90.
   Shima, M., and R. Blacksher, "Improved Microprocessor Interrupt Capability," Electronic Design, 26 Aprile 1978, pp. 96-100.
   Weller, W. J., Practical Microcomputer Programming: the Z80, Northern Technology Books, Evanston, Ill., 1978.
   Winston, A. W., and T. B. Smith, "Use of the Z-80 in Data Collection and Control," IECI '78 Proceedings - Industrial Applications of Microprocessors, 20-22 Marzo 1978, pp. 208-214.

Gli atti del «Meeting» Annuale del Gruppo Elettronica Industriale e Strumentazione di Controllo dell'IEEE in «Industrial Applications of Microprocessors» contengono molti articoli interessanti. I Volumi (a partire dal 1975) sono disponibili da IEEE Service Center, CP Department, 445 Hoes Lane, Piscataway, NJ 08854.

Capitolo 13

DEFINIZIONE DEL PROBLEMA E PROGETTO DEL PROGRAMMA

I COMPITI DELLO SVILUPPO DEL SOFTWARE

Nei capitoli precedenti, abbiamo dedicato la nostra attenzione alla scrittura di brevi programmi in linguaggio assembly. Benché questo costituisca un argomento importante, è solo una piccola parte dello sviluppo del software. Scrivere programmi in linguaggio assembly, sebbene sia il compito principale per chi inizia, diventa presto semplice. Ormai si dovrebbe aver acquisito familiarità con i metodi standard di programmazione in linguaggio assembly sul microprocessore Z80. **I prossimi quattro capitoli descriveranno come formalizzare compiti sotto forma di programmi e come combinare brevi programmi per dar luogo ad un sistema funzionante.**

STADI DELLO SVILUPPO DEL SOFTWARE

Lo sviluppo del software è costituito di più stadi. La fig. 13-1 è un diagramma di flusso del processo di sviluppo del software. **I suoi stadi sono:**

- Definizione del problema.
- Progettazione del programma.
- Codifica.
- Debugging o messa a punto.
- Testing o Collaudo.
- Documentazione.
- Manutenzione e riprogetto.

Ciascuna di queste fasi è importante nella costruzione di un sistema di lavoro. Si noti che la codifica, la scrittura dei programmi in una forma intellegibile per il computer, è soltanto una delle sette fasi.

IMPORTANZA RELATIVA DELLA CODIFICA

Infatti, **la codifica è la fase più facile da definire e da realizzare.** Le regole di scrittura di programmi per il computer sono facili da imparare. Esse variano poco da un computer all'altro, ma le tecniche di base restano immutate. Pochi sono i progetti software che presentano difficoltà per la loro codifica; difatti la codifica non è la parte dello sviluppo software che assorbe maggior tempo. Gli esperti sono del parere che un programmatore sia in grado di scrivere da una a dieci frasi complete di debugging e documentazione. Chiaramente la sola codifica da una fino a dieci frasi è difficilmente lo sforzo completo di un giorno. Nella maggior parte dei progetti software, la codifica occupa meno del 25% del tempo del programmatore.

MISURA DELLO STATO DI AVANZAMENTO NEGLI STADI

Lo sviluppo di misure negli altri stati è difficile. Si può dire che metà del programma è stato scritto, ma si può a malapena affermare che metà degli errori sono stati soppressi o che metà del problema è stato definito. È difficile costruire tabelle di tempo per delle fasi come la progettazione del programma, il debugging ed il collaudo. Inoltre un lavoro incompleto in una fase può comportare problemi molto complessi più tardi. Per esempio, una definizione non accurata del problema od una stesura mediocre del programma possono rendere molto difficili la messa a punto ed il collaudo. Un risparmio di tempo in uno stadio può costare una perdita molto più grossa negli stadi successivi.

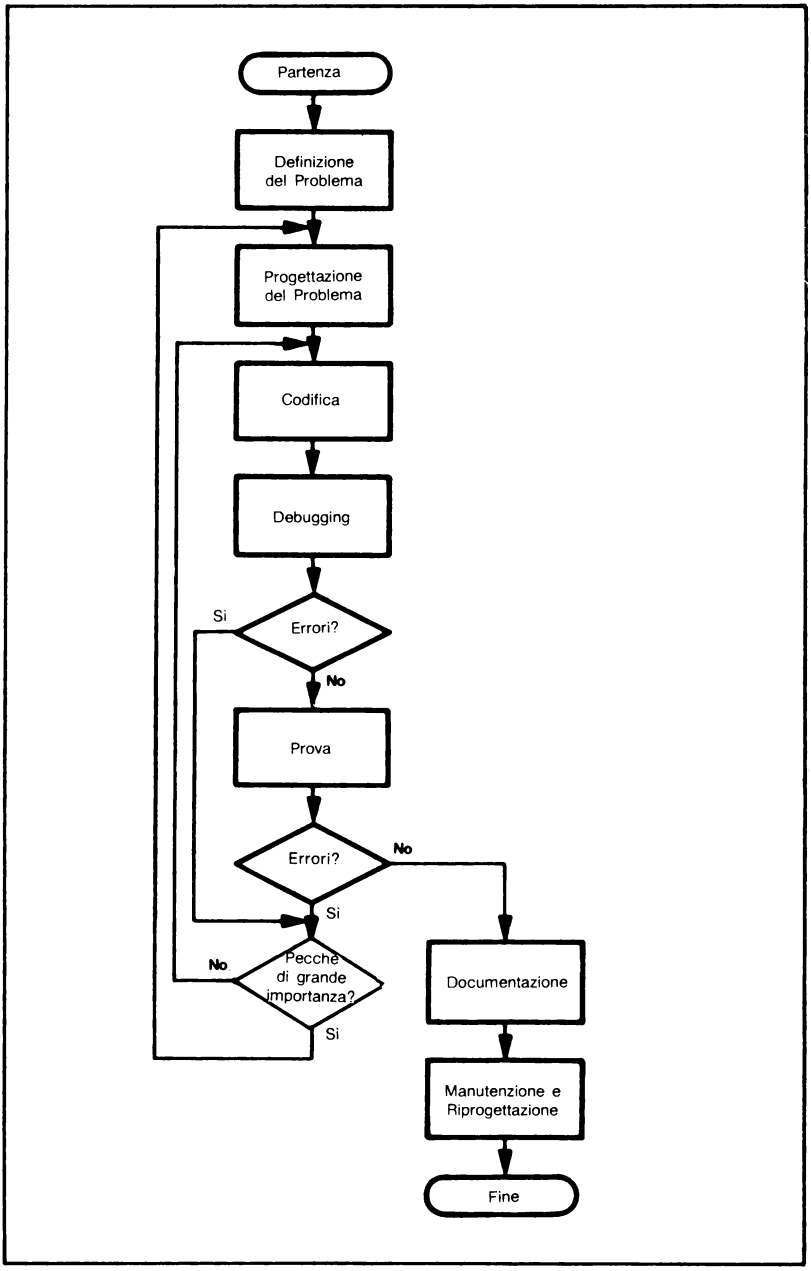


Figura 13-1. Diagramma di Flusso dello Sviluppo Software

DEFINIZIONE DEGLI STADI

DEFINIZIONE DEL PROBLEMA

La definizione del problema è la formulazione del compito in termini di requisiti da imporre al computer. Per esempio, che cosa è necessario per far sì che un computer controlli un utensile o esegua una serie di collaudi elettrici oppure gestisca le comunicazioni tra un controllore centrale ed uno strumento remoto? La definizione del problema richiede che si determinino le forme e le velocità degli ingressi e delle uscite, il contenuto e la velocità dell'elaborazione necessaria ed i tipi di possibili errori, nonché la loro gestione. La definizione del problema comporta una vaga idea di come sia costituito un sistema controllato da computer e definisce i compiti ed i requisiti per il computer.

PROGETTAZIONE DEL PROGRAMMA

La progettazione di un programma è l'abbozzo del programma del calcolatore che eseguirà i compiti che sono stati definiti. Nella fase di progettazione, i compiti sono descritti in maniera tale che possano essere facilmente convertiti in un programma. **Tra le tecniche che risultano utili in questa fase ci sono i diagrammi di flusso, la programmazione strutturata, la programmazione modulare e la stesura 'top-down'.**

CODIFICA

La codifica è la scrittura del programma in una forma che il computer sia in grado di comprendere direttamente o di tradurre. La forma può essere il linguaggio di macchina, il linguaggio assembly o un linguaggio ad alto livello.

DEBUGGING

La messa a punto, o debugging del programma, deve essere eseguita in modo che il programma faccia ciò che è stato specificato in fase di progettazione. In questa fase si ricorre a mezzi come 'breakpoint' (punti di arresto), tracce, simulatori, analizzatori logici ed emulatori in circuito. La fine della fase di messa a punto non è facilmente individuabile, dal momento che non si sa mai quando si è trovato l'ultimo errore.

COLLAUDO

Il collaudo, o testing, riferito anche ad una convalidazione di programma, **consiste nell'assicurare che il programma esegua tutti i compiti di sistema in modo corretto.** Il progettista si serve di simulatori, di esercitatori e di diverse tecniche statistiche per misurare in qualche modo il funzionamento del programma.

DOCUMENTAZIONE

La documentazione è la descrizione del programma in una veste adatta per gli utilizzatori e per il personale di manutenzione. La documentazione inoltre permette al progettista di sviluppare una libreria di programmi in modo che i compiti successivi risulteranno di gran lunga più semplici. Diagrammi di flusso, commenti, mappe di memoria e moduli di libreria sono alcuni dei mezzi usati nella documentazione.

MANUTENZIONE E RIPROGETTAZIONE

La manutenzione e la riprogettazione sono la messa in servizio, le migliorie e l'ampliamento del programma. Chiaramente il progettista deve essere pronto a gestire problemi in campo per apparecchiature basate sul computer. Speciali modi o programmi diagnostici ed altri mezzi di manutenzione possono essere richiesti. Maggiorazioni o ampliamenti del programma si possono rendere necessari per soddisfare nuovi requisiti o gestire nuovi compiti.

Il resto del capitolo considererà soltanto gli stadi di definizione del problema e di stesura del programma. Il capitolo 14 discuterà il debugging ed il collaudo e il capitolo 15 la documentazione, l'ampliamento e la riprogettazione. Tutti gli stadi insieme verranno presentati in qualche semplice esempio di sistema nel capitolo 16.

DEFINIZIONE DEL PROBLEMA

I compiti di un tipico microprocessore richiedono una lunga definizione. Per esempio, cosa deve fare un programma per controllare una bilancia, un registratore di cassa o un generatore di segnale? Chiaramente non è semplice arrivare a definire tutti i compiti coinvolti nel discorso.

DEFINIZIONE DEGLI INGRESSI

Come iniziare la definizione? Il punto più ovvio per incominciare è con gli ingressi. **Si potrebbe iniziare elencando tutti gli ingressi che il computer può ricevere in questa applicazione.**

Esempi di ingressi sono:

- Blocchi di dati da linee di trasmissione.
- Parole di stato da periferiche.
- Dati da convertitori analogico-digitale (A/D).

Allora, ci si può chiedere, relativamente a ciascun ingresso, i seguenti quesiti:

FATTORI IN INGRESSO

1) Qual'è la sua forma; cioè quale segnale il computer riceverà realmente?
2) Quando è disponibile l'ingresso e come fa il processore a riconoscere che esso è disponibile? Il processore deve interrogare l'ingresso con un segnale di strobe? L'ingresso fornisce il proprio clock?
3) Per quanto tempo è disponibile l'ingresso?
4) Quante volte cambia e in che modo il processore si accorge che è cambiato?
5) L'ingresso consiste in una sequenza o in un blocco di dati? È importante l'ordine?
6) Cosa si potrebbe fare se il dato contiene errori? Questi possono comprendere errori di trasmissione, dati non corretti, errori di sequenza, dati supplementari, etc.
7) L'ingresso è in relazione con altri ingressi o uscite?

DEFINIZIONE DELLE USCITE

Il successivo passo da definire è l'uscita. **Si devono elencare tutte le uscite che il computer deve produrre.** Esempi di uscite comprendono:

- Blocchi di dati per linee di trasmissione.
- Parole di controllo per periferiche.
- Dati per convertitori digitale-analogico.

Allora, ci si deve porre le seguenti domande su ogni uscita:

1) Qual'è la sua forma, cioè, quali segnali devono essere prodotti dal computer?
2) Quando deve essere resa disponibile e come la periferica riconosce che essa è disponibile?
3) Quanto tempo deve restare disponibile?
4) Quanto spesso deve cambiare ed in che modo la periferica riconosce che essa è cambiata?
5) C'è una sequenza di uscite? È importante l'ordine?
6) Cosa si potrebbe fare per evitare errori di trasmissione o per avvertire e proteggersi dai guasti di periferiche?
7) Qual'è la relazione tra l'uscita e gli altri ingressi e uscite?

SEZIONE DI ELABORAZIONE

La sezione di elaborazione si estende tra la fase di lettura dei dati di ingresso e la fase di invio dei risultati in uscita. Qui **si deve determinare esattamente in che maniera il computer debba elaborare i dati in ingresso. Le domande sono:**

FATTORI NELL'ELABORAZIONE

1) Qual'è la procedura di base (algoritmo) per trasformare i dati in ingresso nei risultati in uscita?
2) Quali limitazioni di tempo esistono? Queste possono includere le velocità dei dati, i tempi di ritardo, le costanti di tempo dei dispositivi di ingresso e di uscita, etc.
3) Quali limitazione di memoria esistono? Ci sono dei limiti sulla capacità di memoria-programmi o memoria dati, o sulla dimensione dei buffer (tamponi)?
4) Quali programmi o tabelle standard devono essere usati? Quali sono i loro requisiti?
5) Quali casi speciali esistono e in che modo il programma li dovrebbe gestire?
6) Quale deve essere la precisione dei risultati?
7) In che modo il programma dovrebbe gestire gli errori di elaborazione o le condizioni speciali come l'eccedenza della capacità di memoria, l'insufficienza di tale capacità o la perdita di significato?

MANIPOLAZIONE DEGLI ERRORI

Un fattore importante in molte applicazioni è la manipolazione degli errori. Chiaramente, il progettista deve prendere le necessarie precauzioni per premunirsi da errori comuni e per diagnosticare dei malfunzionamenti. **Tra i quesiti che il progettista si deve porre nello stadio di definizione compaiono:**

CONSIDERAZIONI DI ERRORE

1) Quali errori potrebbero presentarsi?
2) Quali errori sono più probabili? Se una persona mette in funzione il sistema, l'errore umano è il più comune. Dopo gli errori umani, gli errori di comunicazione o di trasmissione sono più comuni di quelli meccanici, elettrici, matematici o degli errori del processore.
3) Quali errori non risulteranno immediatamente evidenti per il sistema? Un problema speciale è la presenza di errori che il sistema o l'operatore non sono in grado di riconoscere come sbagliati.
4) In che modo il sistema può proteggersi da errori con una perdita minima di tempo e di dati e cionondimeno essere informato che un errore è avvenuto?
5) Quali errori o malfunzionamenti causano lo stesso comportamento del sistema? Come si possono distinguere questi errori o malfunzionamenti per scopi di diagnostica?
6) Quali errori implicano speciali procedure di sistema? Per esempio, gli errori di parità richiedono una ritrasmissione di dati?

Un'altra domanda è: Come può un tecnico trovare in campo sistematicamente la sorgente di malfunzionamenti senza essere un esperto? Programmi di collaudo incorporati, diagnostiche speciali o signature analysis (analisi di firma) possono essere d'aiuto[1].

FATTORI UMANI

INTERAZIONE DELL'OPERATORE

Molti sistemi basati su microprocessore implicano interazione umana. **I fattori umani devono essere presi in considerazione durante tutto il processo di sviluppo di tali sistemi. Tra i quesiti che il progettista deve porsi compaiono:**

1) Quali procedure di ingresso risultano più naturali per l'operatore umano?
2) È in grado l'operatore di determinare come iniziare, proseguire e terminare le operazioni di ingresso?
3) In che modo l'operatore è informato degli errori di procedura e dei malfunzionamenti dell'impianto?
4) Quali errori l'operatore è più probabile che commetta?
5) In che modo l'operatore riconosce che i dati sono entrati correttamente?
6) I display sono di forma tale che l'operatore li possa facilmente leggere ed interpretare?
7) Il sistema risponde in maniera adeguata per l'operatore?
8) Il sistema è di facile impiego per l'operatore?
9) Esistono delle caratteristiche di guida per un operatore inesperto?
10) Esistono delle opzioni brevi e logiche per l'operatore esperto?
11) L'operatore è sempre in grado di determinare o dare un reset allo stato del sistema dopo interruzioni o distrazioni?

La realizzazione di un sistema per impiego comune presenta delle difficoltà. Il microprocessore può rendere il sistema più potente e flessibile. Comunque il progettista deve aggiungere quel tocco umano che aumenta notevolmente l'utilità, l'attrazione del sistema e la produttività dell'operatore umano.[2]

ESEMPI

Risposta ad un interruttore

DEFINIZIONE DEL SISTEMA INTERRUTTORE E LAMPADA

La Figura 13-2 mostra un semplice sistema in cui l'ingresso proviene da un singolo interruttore SPST (single-pole single-throw: unipolare ad una via) e l'uscita va ad un singolo display LED. In risposta ad una chiusura dell'interruttore, il processore accende il display per un secondo. Questo sistema dovrebbe essere facile da definire.

INGRESSO PER INTERRUTTORE E LAMPADA

Esaminiamo per prima l'ingresso e rispondiamo ad ogni domanda che era stata posta precedentemente:

1) L'ingresso è un singolo bit, che può valere o '0' (interruttore chiuso) o '1' (interruttore aperto).
2) L'ingresso è sempre disponibile e non è necessario che venga richiesto.
3) L'ingresso è disponibile per almeno diversi millisecondi dopo la chiusura.
4) L'ingresso cambierà raramente più di una volta ogni pochi secondi. Il processore deve gestire solo il rimbalzo dell'interruttore. Il processore deve controllare l'interruttore per determinare quando è chiuso.
5) Non c'è sequenza di ingressi.
6) Gli errori evidenti di ingresso sono il guasto dell'interruttore, il guasto nella circuiteria di ingresso e il tentativo, da parte dell'operatore, di chiudere l'interruttore ancor prima che sia trascorsa una sufficiente quantità di tempo. Si discuterà della gestione di questi errori più in avanti.
7) L'ingresso non dipende da altri ingressi o uscite.

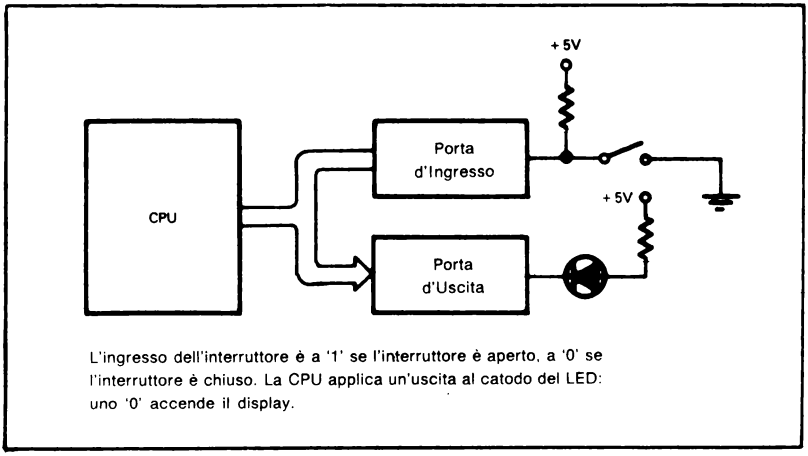


Figura 13-2. Il sistema Interruttore-Display

USCITE PER INTERRUTTORE E LAMPADA

La successiva condizione nella definizione del sistema è esaminare l'uscita. Le risposte alle nostre domande sono:

1) L'uscita è un singolo bit, che vale '0' per accendere il display, '1' per spegnerlo.
2) Non ci sono limitazioni di tempo sull'uscita. Il dispositivo periferico non ha bisogno di essere informato sulla disponibilità di dati.
3) Se il display è un LED, il dato deve essere disponibile soltanto per alcuni millisecondi con una velocità d'impulso di circa 100 volte per secondo. L'osservatore vedrà un display continuamente acceso.
4) I dati devono cambiare (andare in 'off') dopo un secondo.
5) Non ci sono sequenze di uscite.
6) Gli errori possibili in uscita sono il guasto del display e quello del circuito d'uscita.
7) L'uscita dipende soltanto dall'ingresso e dal tempo di commutazione.

La sezione di elaborazione è estremamente semplice. Appena l'ingresso di commutazione diventa una logica '0' la CPU accende la luce (una logica '0') per un secondo. Non esistono limitazioni di tempo.

MANIPOLAZIONE DEGLI ERRORI DI INTERRUTTORE E LAMPADA

Si esaminino adesso i possibili errori e i malfunzionamenti. Questi sono:

- Un'ulteriore chiusura dell'interruttore prima che sia trascorso un secondo.
- Guasto dell'interruttore.
- Guasto del display.
- Guasto del computer.

Sicuramente il primo errore è il più probabile. La soluzione più semplice per il processore è ignorare le chiusure dell'interruttore finché non è trascorso un secondo. Questo breve periodo di non rispondenza risulterà molto apprezzabile ad un operatore umano. Inoltre, se si ignora l'interruttore durante questo periodo, significa che non sono necessari né una circuiteria antirimbalzo né software, poiché il sistema non reagirà in alcun modo al rimbalzo.

Chiaramente, gli ultimi tre guasti possono provocare dei risultati imprevedibili. Il display può essere acceso, spento o cambiare stato solamente. Alcuni possibili modi di isolare i guasti dovrebbero essere:

- Un lamp-test hardware per controllare il display; cioè un pulsante che accenda il display indipendentemente dal processore.
- Una connessione diretta con l'interruttore per rilevare il suo funzionamento.
- Un programma diagnostico che tenga in esercizio i circuiti d'ingresso e d'uscita.

Se sia il display che l'interruttore stanno lavorando, il computer è il responsabile. Un tecnico in campo con un'apparecchiatura adatta può individuare la causa del guasto.

Un Caricatore di memoria basato su interruttori

DEFINIZIONE DI UN CARICATORE DI MEMORIA BASATO SU INTERRUTTORI

La Figura 13-3 mostra un sistema che permette all'utilizzatore di far entrare dati in una locazione di memoria di un microcomputer. Una porta di ingresso DPORT legge i dati da otto interruttori in commutazione. L'altra porta di ingresso CPORT è usata per leggere le informazioni di controllo. Ci sono tre interruttori istantanei: Indirizzo alto, Indirizzo basso e Dati. L'uscita è il valore dell'ultima entrata completa dagli interruttori di dati; otto LED sono usati per il display.

Il sistema richiederà inoltre, naturalmente, diversi resistori, buffer e driver.

Si esamineranno dapprima gli ingressi. Le caratteristiche degli interruttori sono le stesse dell'esempio precedente, comunque, qui esiste una sequenza distinta di ingressi, come segue:

1) L'operatore deve posizionare gli interruttori di dati in conformità degli otto bit più significativi di un indirizzo, poi
2) premere il pulsante Indirizzo alto. I bit dell'indirizzo alto appariranno sui diodi luminosi e il programma interpreterà i dati come il byte alto dell'indirizzo.
3) L'operatore quindi deve posizionare gli interruttori di dati con il valore del byte meno significativo dell'indirizzo e
4) premere il pulsante Indirizzo basso. I bit dell'indirizzo basso appariranno sui diodi luminosi, e il programma considererà i dati come byte basso dell'indirizzo.
5) Infine, l'operatore deve posizionare il dato desiderato negli interruttori di dati e
6) premere il pulsante Dato. Il display mostrerà ora il dato ed il programma lo immagazzinerà in memoria all'indirizzo precedentemente impostato.

L'operatore può ripetere la procedura per far entrare un intero programma. Chiaramente, come in questa situazione semplificata, si avranno molte possibili sequenze da prendere in considerazione. Come proteggersi da sequenze erronee e rendere il sistema facilmente utilizzabile?

L'uscita non è un problema. Dopo ciascun ingresso, il programma invia al display il complemento (dal momento che i display sono attivi bassi) dei bit di ingresso. I dati in uscita restano gli stessi fino alla prossima operazione di ingresso.

La sezione di elaborazione risulta abbastanza semplice. Non ci sono limitazioni di tempo o di memoria. Il programma può proteggersi dai rimbalzi degli interruttori aspettando alcuni millisecondi e deve fornire dati complementati ai display.

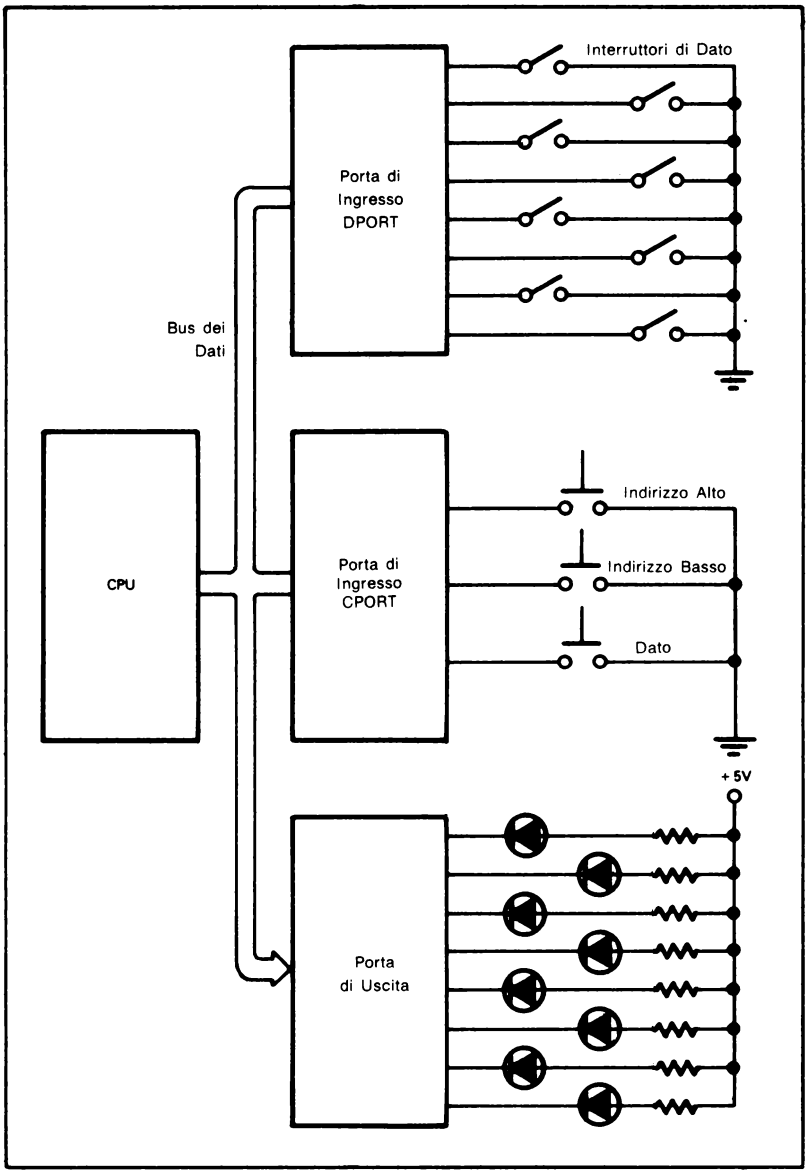


Figura 13-3.II Caricatore di Memoria Basato sull'Interruttore

GESTIONE DEGLI ERRORI DEL CARICATORE DI MEMORIA

Gli errori più probabili sono errori dell'operatore. Questi includono:

- Entrate non corrette.
- Ordine non corretto.
- Entrate incomplete: ad esempio, dimenticanza dei dati.

Il sistema deve essere in grado di gestire questi problemi in modo ragionevole, poiché è certo che avvengono nel funzionamento effettivo.

Il progettista deve inoltre tener conto degli effetti di un guasto della apparecchiatura. Proprio come prima le possibilità difficoltà sono:

- Guasto dell'interruttore.
- Guasto del display.
- Guasto del computer.

In questo sistema comunque si deve fare molta attenzione al modo con il quale questi guasti lo riguardano. Un guasto del computer causerà presumibilmente da parte del sistema un comportamento molto insolito. Un guasto del display non è apprezzabile in maniera immediata: qui una caratteristica di lamp-test permetterà all'operatore di controllare l'operazione. Si noti che si dovrebbe preferire provare ciascun LED separatamente, allo scopo di diagnosticare il caso in cui le linee di uscita siano cortocircuitate tra di loro. In aggiunta, l'operatore non è capace di rilevare immediatamente un guasto dell'interruttore, comunque, l'operatore dovrebbe accorgersene presto e stabilire quale interruttore si è guastato con un procedimento di eliminazione.

Si esaminino alcuni errori possibili d'operatore. Tipici errori saranno:

CORREZIONE DELL'ERRORE DI OPERATORE IN UN CARICATORE DI MEMORIA

- Dati erronei.
- Ordine errato di entrate o interruttori.
- Tentativo di passare alla successiva entrata senza aver completato quella corrente.

L'operatore si accorgerà presumibilmente che i dati sono errati appena appaiono sul display. Qual'è una procedura di ripristino per l'operatore? Qualche opzione è:

1) L'operatore deve completare la procedura di entrata; cioè, inviare l'Indirizzo di ordine basso e il Dato se l'errore si è verificato sull'Indirizzo di ordine alto. Chiaramente questa procedura è dispendiosa e servirebbe soltanto ad annoiare l'operatore.
2) L'operatore può riprendere il processo di entrata ritornando ai passi di entrata dell'indirizzo alto. Questa soluzione si rivela utile se l'errore si è verificato sull'Indirizzo Alto, ma obbliga l'operatore a far entrare di nuovo i dati precedenti se l'errore si è verificato nella fase Indirizzo Basso o Dato.
3) L'operatore può far entrare qualsiasi parte della sequenza ad un qualunque istante posizionando semplicemente gli interruttori Dato con il desiderato dato e premendo il corrispondente pulsante. Questa procedura permette all'operatore di apportare correzioni in un qualsiasi punto della sequenza.

Questo tipo di procedura sarebbe sempre da preferire ad una non in grado di permettere la correzione immediata dell'errore, con una molteplicità di passi finali, o con entrata di dati nel sistema senza la possibilità da parte dell'operatore di un controllo finale. Qualunque complicazione aggiunta in hardware o in software verrà giustificata da un aumento dell'efficienza d'operatore. Si dovrebbe preferire sempre di lasciare al microcomputer il compito di svolgere il lavoro tedioso e quello di riconoscere sequenze arbitrarie; esso non si stanca mai né dimentica che cosa era nel manuale di funzionamento.

Una ulteriore caratteristica utile dovrebbe consistere nelle luci di stato che definirebbero il significato del display. Tre luci di stato, contrassegnate «Indirizzo Alto», «Indirizzo Basso» e «Dato», permetterebbero all'operatore di sapere che cosa è entrato senza bisogno di ricordare quale pulsante aveva premuto. Il processore dovrebbe controllare la sequenza, ma la complicazione aggiunta al software semplificherebbe il compito dell'operatore. Chiaramente, tre separate serie di display e in più la possibilità di esaminare una locazione di memoria dovrebbero risultare altrettanto utili all'operatore.

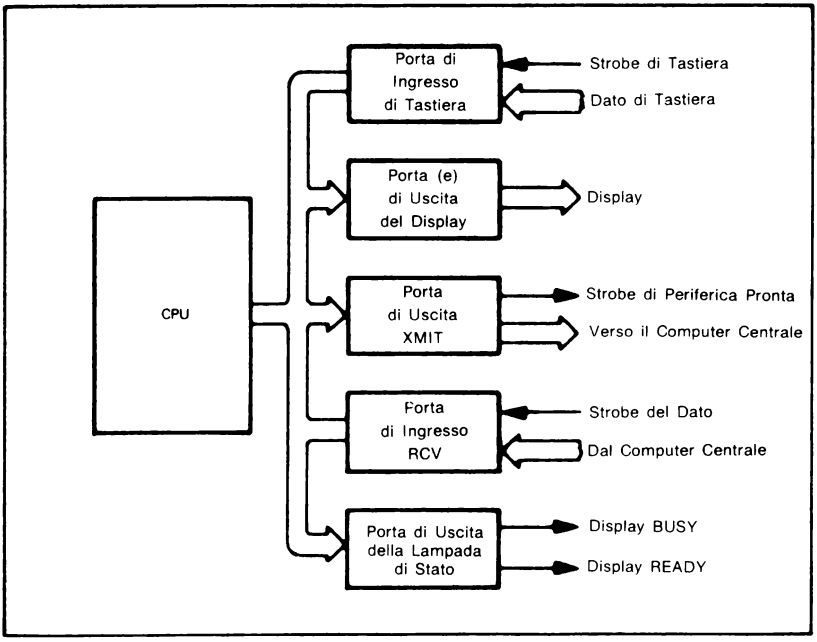


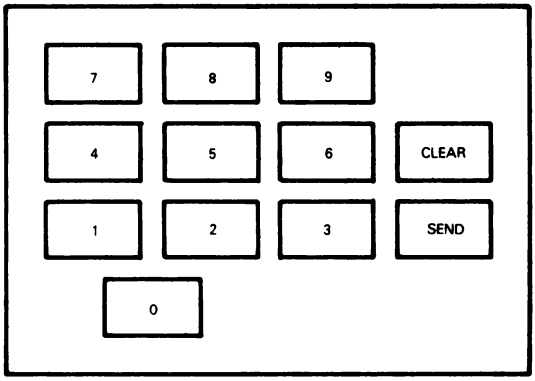
Figura 13-4. Diagramma a Blocchi di un Terminale di Verifica

Si dovrebbe notare che, sebbene si è data enfasi all'interazione umana, l'interazione della macchina o del sistema ha molte caratteristiche simili. Il microprocessore compierebbe il lavoro. Se complicando il compito del microprocessore si rende semplice il ripristino dell'errore ed evidenti le cause di guasto, l'intero sistema lavorerà in maniera migliore e risulterà più facile la manutenzione. Si noti che non si dovrebbe aspettare fino a dopo che il software abbia terminato di esaminare l'uso e la manutenzione del sistema: invece, si potrebbe includere questi fattori nella fase di definizione del problema.

Un Terminale di Verifica

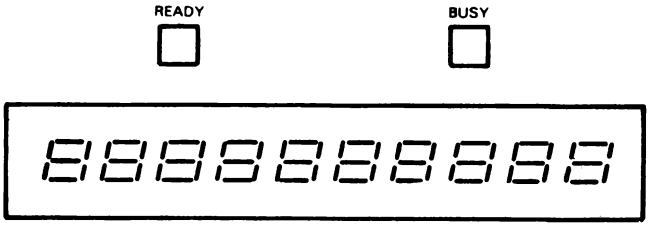
DEFINIZIONE DI UN TERMINALE DI VERIFICA

La figura 13-4 è uno schema a blocchi di un semplice terminale di verifica-di-credito. Una porta d'ingresso attinge dati da una Tastiera (vedi figura 13-5); l'altra porta d'ingresso accetta dati di verifica da una linea di trasmissione. Una porta d'uscita invia il numero di carta di credito al computer centrale. Una terza porta d'uscita accende una luce quando il terminale è pronto ad accettare una richiesta e un'altra luce quando l'operatore invia l'informazione. La luce «Occupato» si spegne quando la risposta ritorna. Chiaramente, l'ingresso e l'uscita del dato risulterà più complessa del caso precedente, sebbene la procedura sia ancora semplice.



I tasti digitali permettono ingressi digitali.
CLEAR annulla totalmente l'entrata.
SEND trasmette l'entrata al computer centrale.

Figura 13-5. Tastiera del Terminale di Verifica



Il display consiste di dieci display a 7 segmenti, che possono essere messi in multiplex, controllati da un registro a scorrimento, o indirizzati separatamente. Due lampade aggiuntive, READY e BUSY, sono pure presenti.

Figura 13-6. Display del Terminale di Verifica

Display addizionali possono risultare utili per dare più enfasi al significato della risposta. Molti terminali usano una luce verde per il «Sì» ed una rossa per il «No» ed una gialla per «Consulenza Organizzazione Memoria». Notare che le luci dovranno inoltre essere chiaramente contrassegnate con il loro significato per tenere conto di un operatore daltonico.

INGRESSI DEL TERMINALE DI VERIFICA

Si guardi dapprima l'ingresso della tastiera. Questa è differente dall'ingresso a interruttore, naturalmente, poiché la CPU deve poter distinguere il nuovo dato. **Si assumerà che ciascun tasto di chiusura fornisca un unico codice esadecimale (si può codificare ciascuno dei 12 tasti con un digit) ed uno strobe. Il programma dovrà riconoscere lo strobe e andare a prendere il numero esadecimale che identifica il Tasto.** Esiste una limitazione di tempo, poiché il programma non può perdere nessun dato o strobe. La limitazione non è severa, dal momento che le entrate della Tastiera saranno almeno della durata di diversi millisecondi ciascuna.

L'ingresso di trasmissione consiste, in maniera del tutto simile, in una serie di caratteri, ciascuno identificato da uno strobe (forse da un UART). Il programma dovrà riconoscere ciascun strobe e andare a prendere il carattere. I dati inviati attraverso le linee di Trasmissione sono normalmente organizzati a messaggi. Un possibile formato di messaggio è:

- Carattere di introduzione o guida (header).
- Indirizzo della destinazione del terminale.
- Codificato sì o no.
- Caratteri di terminazione o trascinatore (trailer).

Il terminale controllerà l'header, leggerà l'indirizzo di destinazione e vedrà se il messaggio è per sè. Se il messaggio è per il terminale, il terminale accetta i dati. L'indirizzo potrebbe essere (e spesso è) calblato fisso nel terminale, cosicché il terminale riceve solo messaggi destinatigli. Questa impostazione semplifica il software a scapito della flessibilità.

USCITE VERSO IL TERMINALE DI VERIFICA

L'uscita è ancora più complessa che nei precedenti esempi. Se i display sono multiplexed, il processore non deve soltanto inviare i dati alla porta del display ma deve anche inviare il dato in direzione di un particolare display. Si avrà bisogno o di una porta separata di controllo o di un contatore e di un decodificatore per gestire questo problema. Si faccia attenzione al fatto che i controlli hardware di spaziatura possono rendere blank i primi zero per tutto il tempo che il primo digit, in un numero multi-digit, non è mai zero. Anche il software può gestire questo compito. Le limitazioni di tempo includono la lunghezza d'impulso e la frequenza richiesta per dare una visualizzazione continua all'operatore.

Le uscite di comunicazione saranno costituite da una serie di caratteri con un formato particolare. Un programma inoltre dovrà considerare il tempo richiesto tra caratteri. Il formato possibile per un messaggio di uscita è:

- Header.
- Indirizzo Terminale.
- Numero della carta di credito.
- Trailer.

Un computer di comunicazione centrale può interrogare i terminali, controllando i dati pronti per essere inviati.

La procedura in questo sistema coinvolge molti nuovi compiti, come:

- Identificazione dei tasti di controllo dal numero ed esecuzione delle azioni adeguate.
- Aggiunta dell'header, dell'indirizzo Terminale, e del trailer al messaggio uscente.
- Riconoscimento dell'header e del trailer nel messaggio di ritorno.
- Controllo dell'indirizzo del terminale entrante.

Si noti che nessuno dei compiti comporta dell'aritmetica complessa o delle limitazioni severe di tempo o di memoria.

MANIPOLAZIONE DEGLI ERRORI DEL TERMINALE DI VERIFICA

Il numero di errori possibili in questo sistema è naturalmente più grande che nei precedenti esempi. Prendiamo in considerazione dapprima i possibili errori d'operatore. Questi includono:

- Entrata non corretta del numero della carta di credito.
- Tentativo di inviare un numero di carta di credito incompleto.
- Tentativo di inviare un numero mentre il computer centrale ne sta elaborando un altro.
- Azzeramento di entrate non esistenti.

Alcuni di questi errori possono essere facilmente gestiti strutturando in modo corretto il programma. Per esempio il programma non potrebbe accettare il tasto 'Invia' (Send) fino a che il numero della carta di credito non è completamente entrato e dovrebbe ignorare qualunque entrata addizionale da Tastiera fino a che non ritorna indietro dal computer centrale la risposta. Si noti che l'operatore si accorgerà che l'entrata non è stata inviata, poiché la luce «Occupato» (Busy) non si sarà accesa. L'operatore inoltre si accorgerà quando la tastiera è stata bloccata — (il programma sta ignorando le entrate da Tastiera) —, dal momento che le entrate non compaiono sul display e la luce 'Pronto' (Ready) sarà spenta.

CORREZIONE DEGLI ERRORI DI TASTIERA

Le entrate non corrette sono un problema evidente. Se l'operatore riconosce un errore egli può usare il tasto Azzera (Clear) per eseguire delle correzioni. L'operatore probabilmente troverebbe ciò più conveniente che non l'avere a disposizione due tasti Azzera, uno che annulla il Tasto più recente e un altro che annulla l'intera entrata. Questo terrebbe conto di entrambe le situazioni in cui l'operatore riconosce l'errore o immediatamente o più avanti nella procedura. L'operatore sarebbe in grado di correggere gli errori immediatamente e dovrebbe ripetere il minor numero di tasti possibile. Comunque l'operatore farà un certo numero di errori senza riconoscerli. La maggior parte dei numeri di carta di credito include un digit di autocontrollo; il terminale potrebbe controllare il numero prima di consentirgli di essere inviato al computer centrale. Questo passo salvaguarderebbe il computer centrale dallo sprecare del tempo prezioso di elaborazione per il controllo del numero.

Questo richiede, comunque, che il terminale abbia modo di informare l'operatore dell'errore, forse illuminando uno dei display o fornendo un altro speciale indicatore che sicuramente l'operatore noti.

Ancora un altro problema è il modo in cui l'operatore riesca a capire che una entrata è stata persa o elaborata in maniera non corretta. Alcuni terminali si sbloccano semplicemente dopo un tempo massimo di ritardo. L'operatore si accorge che la luce 'Occupato' si è spenta senza che una risposta sia stata ricevuta. L'operatore quindi pensa di tentare di entrare ancora. Dopo uno o due tentativi, l'operatore dovrebbe riferire il guasto al personale di supervisione.

Molti guasti di apparecchiatura sono del resto possibili. Accanto ai display, tastiera e processore, ci sono ora problemi di errori o guasti di comunicazioni e guasti del computer centrale.

La trasmissione dati dovrà probabilmente comprendere le procedure di controllo e correzione dell'errore. Alcune possibilità sono:

CORREZIONE DEGLI ERRORI DI TRASMISSIONE

1) La parità permette una facile rivelazione dell'errore, ma non un meccanismo di correzione. Il ricevitore avrà bisogno in qualche modo di richiedere la ritrasmissione e il trasmettitore dovrà salvare una copia del dato finché non venga riconosciuta la ricezione esatta. La parità è veramente semplice da implementare.
2) Dei messaggi corti possono servirsi di schemi più accurati. Ad esempio, la risposta si/no al terminale potrebbe essere codificata in modo tale da fornire una capacità di rilevare e correggere l'errore.
3) Un riconoscimento ed un numero limitato di tentativi potrebbero innescare un indicatore che sia in grado di informare l'operatore di un guasto di comunicazione (incapacità di trasferire un messaggio senza errori) o del computer centrale (mancata risposta al messaggio entro un certo periodo di tempo). Un tale schema, combinato con un lamp-test, permetterebbe diagnosi di semplici guasti.

Un indicatore di guasto nel computer centrale o nella comunicazione «sbloccherebbe» inoltre il terminale, cioè, gli permetterebbe di accettare un'altra entrata. Questo è necessario se il terminale non accetterà entrate mentre è in fase di attuazione una verifica. Certe entrate potrebbero essere riservate a diagnostiche; cioè, certi numeri di carta di credito verrebbero usati per controllare l'operazione interna del terminale e controllare i display.

ANALISI DELLA DEFINIZIONE DEL PROBLEMA

La definizione del problema è una parte importante dello sviluppo software così pure come di un qualsiasi altro compito sistemistico (ingegneristico). Si noti che essa non richiede alcuna programmazione o conoscenza del computer; piuttosto, si basa su una comprensione del sistema e su un profondo giudizio ingegneristico. I microprocessori possono offrire una flessibilità che il progettista può impiegare per fornire un campo di caratteristiche precedentemente non disponibili.

La definizione del problema è indipendente da qualunque computer particolare, linguaggio per computer, o sistema di sviluppo. Comunque essa fornirebbe dei modi di condotta come per quale tipo di computer o velocità di computer l'applicazione richiederà, e quale tipo di compromesso hardware/software il progettista può svolgere. Lo stadio di definizione del problema è infatti indipendente dal fatto che un computer venga o no impiegato completamente, sebbene una conoscenza delle capacità del computer possa aiutare il progettista nel suggerire possibili implementazioni di procedure.

PROGETTAZIONE DEL PROGRAMMA

La progettazione del programma è la fase in cui la definizione del problema è formulata come un programma. Se il programma è piccolo e semplice, questa fase può comportare poco più della scrittura di un diagramma di flusso di una pagina. Se il programma è più esteso e più complesso, il progettista dovrebbe prendere in esame metodi più elaborati.

Si discuterà il diagramma di flusso, la programmazione modulare, la programmazione strutturata e la progettazione «top-down». Si tenterà di indicare il ragionamento che sta alla base di questi metodi ed i loro vantaggi e svantaggi. Comunque non si difenderà alcun metodo particolare dal momento che non risulta evidente la prevalenza continua di uno su tutti gli altri. Si dovrebbe ricordare che lo scopo è dar luogo ad un buon sistema di lavoro, non seguire fedelmente i dogmi di una metodologia o di un'altra.

PRINCIPI BASILARI DELLA PROGETTAZIONE DI UN PROGRAMMA

Tutte le metodologie, comunque, **possiedono dei semplici principi comuni.** Molti di questi sono quegli stessi principi che ritroviamo applicati ad ogni tipo di progetto, **come:**

1) Procedere per piccoli passi. Non tentare di fare troppo alla volta.

2) Dividere grossi lavori in piccoli, separare logicamente i compiti. Costruire dei sottocompiti il più possibile indipendenti gli uni dagli altri, cosicché possano essere collaudati separatamente e le modifiche possono avvenire in uno senza influenzare gli altri.

3) Prendere il flusso di controllo il più semplice possibile in modo da renderlo più comodo nella ricerca degli errori.

4) Usare il più possibile descrizioni grafiche o illustrate. Esse sono più semplici da visualizzare che non le descrizioni scritte. Questo è il maggior vantaggio dei diagrammi di flusso.

5) Innanzitutto dare maggiore importanza alla chiarezza e alla semplicità. Si può migliorare la realizzazione (se necessario) una volta che il sistema sia funzionante.

6) Procedere in modo sistematico e completo. Usare liste di controllo e procedure standard.

7) Non tentare la sorte. Non ricorrere a metodi del cui impiego non si è sicuri o usarli con molta attenzione. Stare attenti a situazioni che potrebbero essere causa di confusione e chiarirle il più presto possibile.

8) Tenere presente la messa a punto del sistema, il collaudo e la manutenzione. Pianificare il tutto in vista di questi stadi successivi.

9) Servirsi di metodi e di terminologie semplici e consistenti. La ripetitività nella stesura del programma non fa male, la complessità non è una virtù.

10) Formulare completamente la stesura prima di iniziare la codifica. Non lasciarsi tentare dall'iniziare a scrivere istruzioni: non ha molto senso fare liste parziali o collocare schede di circuiti prima di conoscere con esattezza che cosa ci sarà nel sistema.

11) Porre particolare attenzione ai fattori che possono cambiare. Fare l'implementazione delle probabili modifiche il più semplicemente possibile.

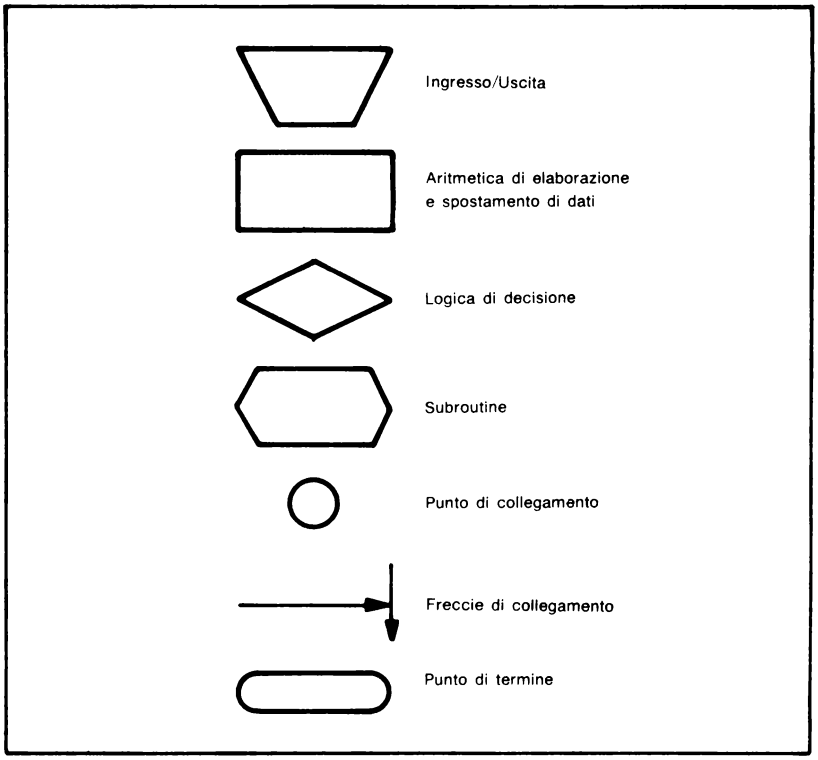


Figura 13-7. Simboli del Diagramma di Flusso Standard

DIAGRAMMI DI FLUSSO

Il diagramma di flusso è certamente il più conosciuto di tutti i metodi di stesura di programma. I libri di programmazione descrivono come i programmatori innanzitutto traccino completi diagrammi di flusso e poi inizino a scrivere il programma effettivo. Infatti, alcuni programmatori hanno sempre lavorato in questo modo, ed il diagramma di flusso spesso non è stato altro che un gioco o un fastidio per i programmatori più che un metodo di progetto. Si cercherà di descrivere i vantaggi e gli svantaggi dei diagrammi e di mostrare il ruolo di questa tecnica nella stesura del programma.

VANTAGGI DEI DIAGRAMMI DI FLUSSO

Il vantaggio fondamentale del diagramma di flusso è che consiste in una rappresentazione. Tali rappresentazioni risultano a chi le guarda più significative delle descrizioni scritte. Il progettista può visualizzare il sistema completo e vedere le relazioni intercorrenti tra le varie parti. Errori logici ed incompatibilità spesso sono di provenienza esterna al posto di quelli nascosti in un foglio stampato. **Il diagramma di flusso, al meglio, è una rappresentazione dell'intero sistema.**

Alcuni dei vantaggi peculiari dei diagrammi di flusso sono:

1) Esistono simboli standard (vedere Figura 13-7) di modo che i moduli del diagramma siano facilmente riconoscibili.
2) I diagrammi di flusso possono essere capiti da persone senza esperienza precedente di programmazione.
3) I diagrammi di flusso possono essere impiegati per modularizzare l'intero progetto in sotto-compiti. Essi possono poi essere esaminati per misurare l'avanzamento globale.
4) I diagrammi di flusso mostrano il susseguirsi delle operazioni e pertanto possono aiutare ad individuare la sorgente di errori.
5) Stendere un diagramma di flusso è molto utile anche in altre aree al di là della programmazione.
6) Ci sono molti mezzi a disposizione per aiutare la stesura di un diagramma di flusso, che includono mascherine per programmatori e configurazioni per disegno automatico.

SVANTAGGI DEI DIAGRAMMI DI FLUSSO

Questi vantaggi sono tutti importanti. Non esistono dubbi sul largo impiego che si continuerà a fare dei diagrammi di flusso. **Ma si potrebbero notare alcuni svantaggi dei diagrammi di flusso, impiegato come metodo di stesura di un programma, per esempio:**

1) I diagrammi di flusso sono difficili da stendere, disegnare o cambiare del tutto, eccetto nelle situazioni più semplici.
2) Non esistono dei modi facili di debugging (messa a punto) di un diagramma di flusso o di collaudo.
3) I diagrammi tendono a diventare confusi. I progettisti trovano complicato bilanciare la quantità di dettagli necessari a rendere il diagramma utile e quella che rende il diagramma un po' migliore del listing di programma.
4) I diagrammi di flusso mostrano soltanto l'organizzazione del programma. Essi non mostrano l'organizzazione dei dati o la struttura dei moduli di ingresso/uscita.
5) I diagrammi di flusso non sono d'aiuto nell'hardware o nei problemi di temporizzazione né danno suggerimenti di dove questi problemi si potrebbero verificare.
6) I diagrammi di flusso permettono progetti altamente non strutturati. Il tracciare a ritroso linee e righe come anche tutti i cicli sopra la carta sono in antitesi ai principi di un progetto ben strutturato.

Dunque, **il diagramma di flusso è una tecnica utile** che non si dovrebbe cercare di estendere troppo. **I diagrammi di flusso sono utili come documentazione di programma, dal momento che hanno forme standard e sono comprensibili ai non programmatori.** Come mezzo di progetto, **comunque,** i diagrammi di flusso costituiscono molto di più che un abbozzo di partenza; **il programmatore non può mettere a punto un diagramma di flusso dettagliato** ed il diagramma di flusso è spesso molto più difficile da stendere del programma stesso.

ESEMPI

Risposta ad un Interruttore

DIAGRAMMA DI FLUSSO DI UN SISTEMA CON INTERRUTTORE E LAMPADA

Questo semplice compito, in cui un singolo interruttore accende una luce per un secondo, è facilmente rappresentabile con un diagramma di flusso. Infatti, tali compiti rientrano in certi tipici esempi adottati da testi sui diagrammi di flusso, sebbene essi siano una piccola parte della maggior parte dei sistemi. La struttura dei dati qui è così semplice che può venire trascurata con tutta sicurezza.

La Figura 13-8 è il diagramma di flusso. C'è un po' di difficoltà nel decidere l'ammontare di dettaglio richiesto. Il diagramma di flusso dà una rappresentazione diretta della procedura, che chiunque potrebbe capire.

Si noti che i diagrammi di flusso più utili possono ignorare variabili di programma e chiedere domande direttamente. Naturalmente, sono spesso necessari dei compromessi. **Risultano spesso utili due versioni del diagramma di flusso — una versione generale in linguaggio da profano che sarà utile ai non-programmatori, e una versione per programmatori in termini di variabili del programma, che sarà utile agli altri programmatori.**

DIAGRAMMA DI FLUSSO DI DATI

Un terzo tipo di diagramma di flusso, un diagramma di flusso di dati, può essere pure utile. Questo serve come riferimento per altri diagrammi, dal momento che mostra come il programma gestisce un particolare tipo di dati. Diagrammi di flusso ordinari mostrano il modo di procedere di un programma, la gestione di differenti tipi di dato in punti diversi.

I diagrammi di flusso di dati, d'altro canto mostrano come particolari tipi di dati si muovono nel sistema, passando da una parte all'altra del programma. Tali diagrammi di flusso risultano molto utili nella messa a punto e nella manutenzione, dal momento che si evidenziano sempre più degli errori appena un particolare tipo di dati è gestito in maniera non corretta.

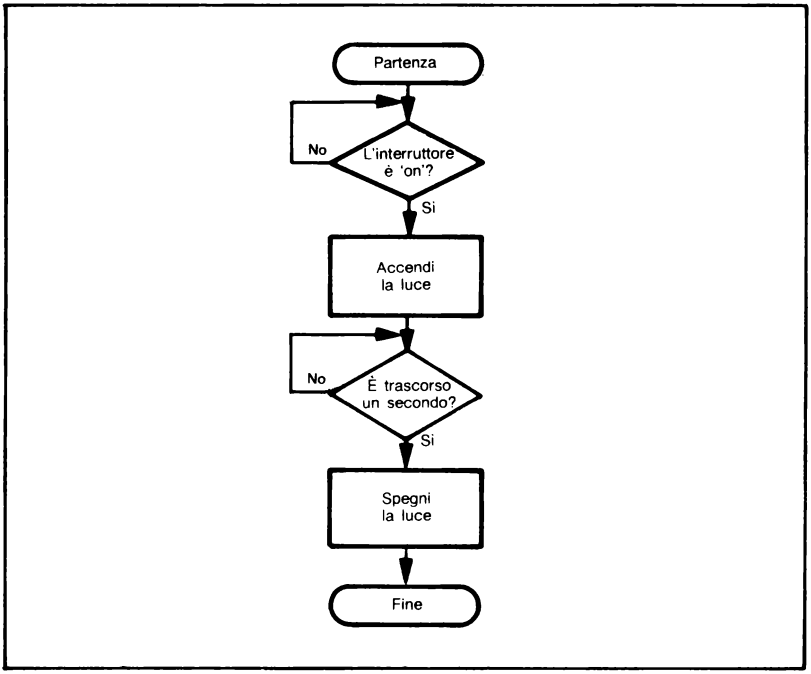


Figura 13-8. Diagramma di Flusso della Risposta di un Secondo ad un Interruttore

Il Caricatore di Memoria Basato su Interruttore

DIAGRAMMA DI FLUSSO DEL CARICATORE DI MEMORIA BASATO SU INTERRUTTORI

Questo sistema (vedere Figura 13-3) è considerevolmente più complesso del precedente esempio e include un numero maggiore di decisioni. **Il diagramma di flusso (vedere Figura 13-9) è più difficile da scrivere e non direttamente come prima.** In questo esempio, si affronta il problema che non esiste maniera di mettere a punto o collaudare il diagramma di flusso.

Il diagramma di flusso in Figura 13-9 comprende le migliorie che vengono suggerite come parte della definizione del problema. Chiaramente, **questo diagramma di flusso comincia ad essere confuso ed a perdere i suoi vantaggi rispetto ad una descrizione scritta.** Aggiungendo altre caratteristiche che definiscono il significato dell'entrata con luci di stato e permettono all'operatore di controllare le entrate dopo completamento, si dovrebbe rendere il diagramma di flusso molto più complesso. Scrivere il diagramma di flusso completo dal niente potrebbe diventare rapidamente un compito notevole. Comunque, una volta che il programma è stato scritto, il diagramma di flusso è utile come documentazione.

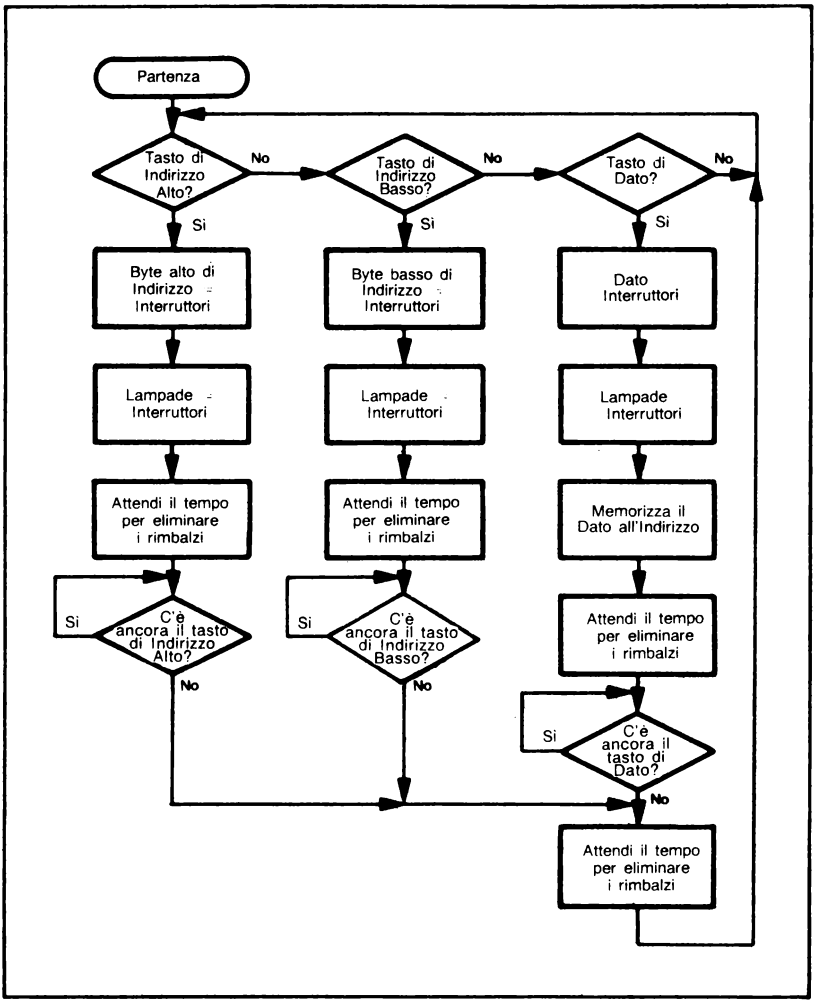


Figura 13-9. Diagramma di Flusso di un Caricatore di Memoria Basato sull'Interruttore

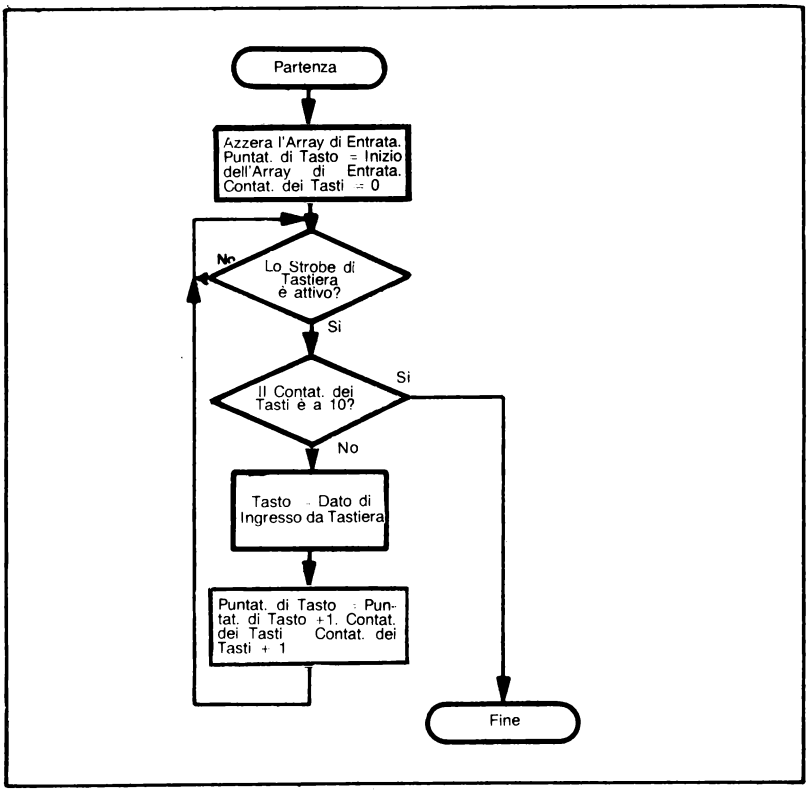


Figura 13-10. Diagramma di Flusso del Processo d'Ingresso da Tastiera

Il Terminale di Verifica di Credito

DIAGRAMMA DI FLUSSO DELLA VERIFICA DI CREDITO

SEZIONI DI DIAGRAMMA DI FLUSSO

In questa applicazione (vedere Figure 13-4 a 13-6), il diagramma di flusso sarà più complesso di quello per il caso del caricatore di memoria basato su interruttori. Qui, **l'idea migliore è di realizzare sezioni separate di diagramma in modo che il diagramma di flusso resta maneggevole.** Comunque, la presenza di strutture dati (come nel display multi-digit e nei messaggi) renderanno più grande il passo dal diagramma di flusso al programma.

Si dia uno sguardo ad alcune sezioni. **La Figura 13-10 mostra il procedimento di entrata da tastiera per tasti numerici.** Il programma deve catturare il dato dopo ogni strobe e piazzare il digit nell'array di display, se c'è posto. Se ci sono già dieci digit nell'array, il programma si limita ad ignorare semplicemente l'entrata.

Il programma effettivo dovrà gestire i display contemporaneamente. Si noti che sia l'hardware che il software devono rendere inattivo lo strobe di tastiera dopo che il processore ha letto un digit.

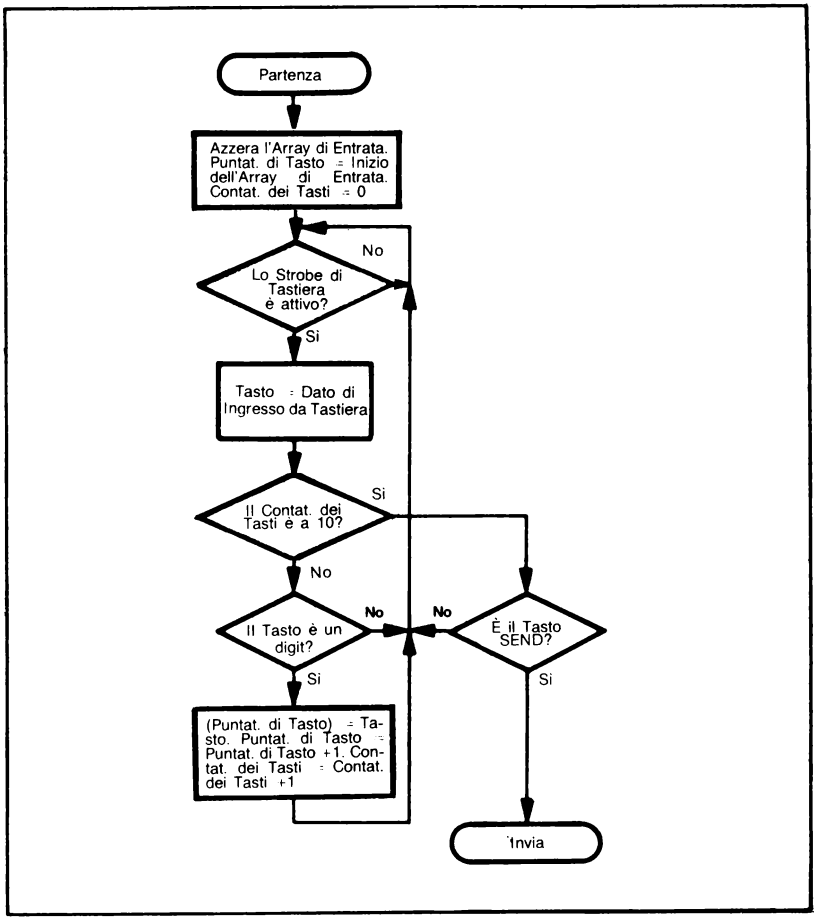


Figura 13-11. Diagramma di Flusso del Processo d'Ingresso da Tastiera con il Tasto SEND

La Figura 13-11 aggiunge il tasto 'Invia'. Questo tasto, naturalmente, è opzionale. Il terminale potrebbe inviare già il dato appena l'operatore fa entrare un numero completo. Comunque, quella procedura non dovrebbe permettere all'operatore di controllare l'intera entrata. Il diagramma di flusso con il tasto 'Invia' è più complesso poichè ha due alternative.

1) Se l'operatore non ha caricato 10 digit, il programma deve ignorare il tasto 'Invia' e piazzare un altro tasto in entrata.
2) Se l'operatore ha caricato 10 digit, il programma deve rispondere al tasto 'Invia' trasferendo il controllo alla routine Invia (Send) e ignorare tutti gli altri tasti.

Si noti che il diagramma di flusso è diventato molto più difficile da organizzare e da seguire. Inoltre non esiste un modo ovvio di controllare il diagramma di flusso.

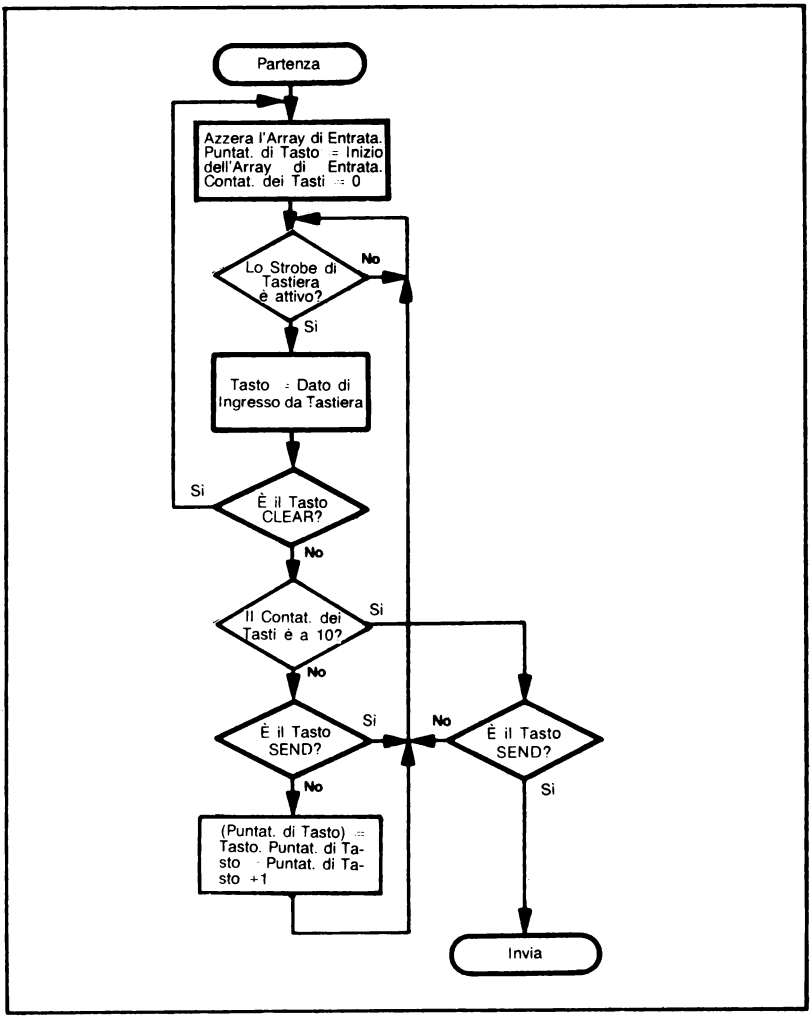


Figura 13-12. Diagramma di Flusso del Processo d'Ingresso da Tastiera con i Tasti di Funzione

La Figura 13-12 mostra il diagramma di flusso del processo di entrata da tastiera con tutti i tasti di funzione. In questo esempio, il flusso di controllo non è semplice. Chiaramente, alcune descrizioni scritte risultano necessarie. L'organizzazione e la disposizione di diagrammi di flusso complessi richiedono una pianificazione accurata. Si è seguito il procedimento di aggiungere caratteristiche al diagramma di flusso una alla volta, ma questo comporta ancora un ammontare non indifferente di ricompilazione. Si dovrebbe inoltre tener presente che attraverso la procedura di entrata da tastiera, il programma deve inoltre rinfrescare i display se sono in multiplex e non controllati da registri a scorrimento o da altro hardware.

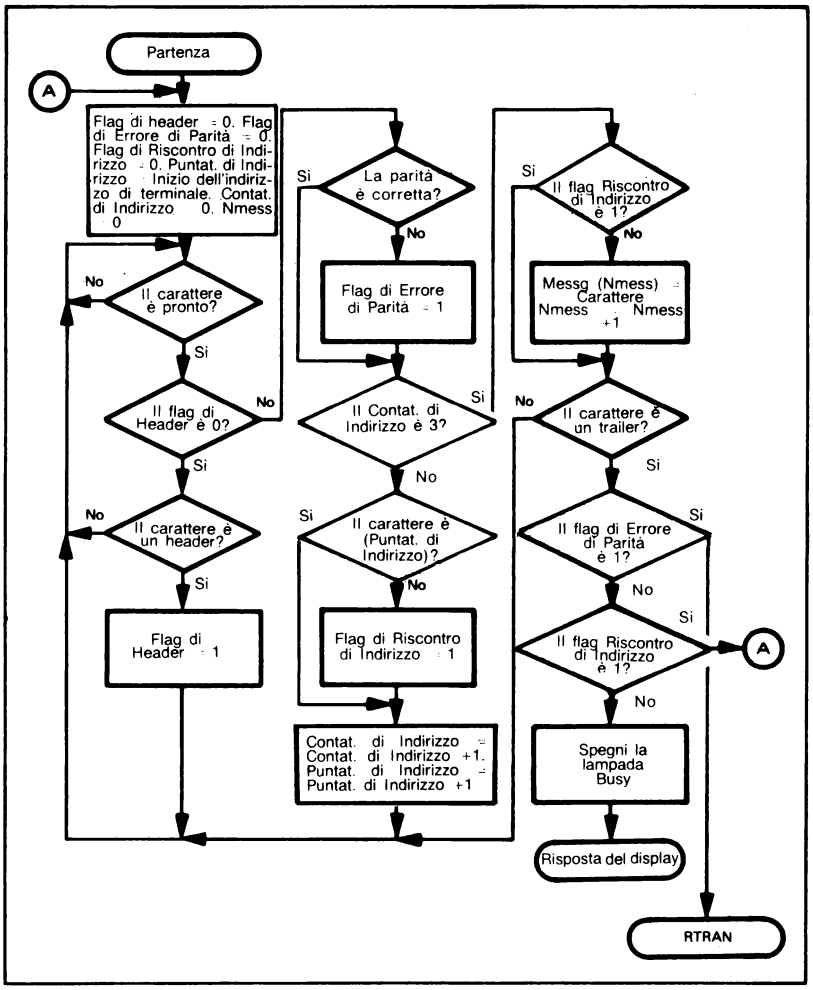


Figura 13-13. Diagramma di Flusso della Routine di Ricezione

La Figura 13-13 è il diagramma di flusso di una routine di ricezione. Si assume che la conversione seriale/parallelo e la rilevazione dell'errore siano eseguite via hardware (cioè da un UART). Il processore deve:

1) Cercare l'header (si assume che sia un singolo carattere).
2) Leggere l'indirizzo di destinazione (si assume che sia lungo tre caratteri) e vedere se il messaggio ha significato per questo terminale; cioè se i tre caratteri sono in accordo con l'indirizzo del terminale.
3) Attendere il carattere di trailer.
4) Se il messaggio ha significato per il terminale, spegnere la luce 'Occupato' e andare alla routine di 'Rispondi al Display'.
5) Nel caso di alcuni errori, richiedere la ritrasmissione andando nella routine RTRAN.

Questa routine include un largo numero di decisioni e il diagramma di flusso non si presenterà né semplice né ovvio.

Chiaramente, si è andati molto lontani dal diagramma di flusso semplice (Figura 13-8) del primo esempio. Una serie completa di diagrammi di flusso per un terminale di transazione sarebbe un compito maggiore. Ciò consisterebbe di diversi fogli di programmi in relazione tra di loro con logica complessa e richiederebbe un enorme sforzo. Un tale sforzo dovrebbe rivelarsi così difficile già nello scrivere un programma preliminare, senza essere altrettanto utile, dal momento che non è detto che sia controllabile sul computer.

PROGRAMMAZIONE MODULARE

Una volta che i programmi diventano lunghi e complessi, il diagramma di flusso non risulta più un mezzo di progetto soddisfacente.

Comunque, la definizione del problema e il diagramma di flusso possono dare un'idea di come suddividere il programma in sotto-compiti accettabili. **La divisione dell'intero programma in sotto-compiti o moduli è detto «programmazione modulare».** Chiaramente, la maggior parte dei programmi presentati nei capitoli precedenti dovrebbero essere tipici moduli in un più ampio programma di sistema. **I problemi che il progettista affronta in una programmazione modulare sono il modo di suddividere il programma in moduli e il modo di connetterli insieme.**

VANTAGGI DELLA PROGRAMMAZIONE MODULARE

I vantaggi della programmazione modulare sono:

1) La scrittura, il debugging ed il collaudo di un modulo singolo sono più facili rispetto ad un intero programma.
2) Un modulo può risultare utile probabilmente in diversi posti ed in altri programmi, particolarmente se è ragionevolmente generale ed esegue un compito comune. Si può costruire una libreria di moduli standard.
3) La programmazione modulare permette al programmatore di distribuire i compiti e servirsi dei programmi precedentemente scritti.
4) Modifiche possono essere introdotte in un modulo piuttosto che in un intero sistema.
5) Gli errori possono essere spesso isolati e poi attribuiti ad un singolo modulo.
6) La programmazione modulare dà un'idea di quanto progresso si è fatto e di quanto lavoro rimane.

SVANTAGGI DELLA PROGRAMMAZIONE MODULARE

L'idea della programmazione modulare è di una chiarezza tale che i suoi svantaggi vengono spesso trascurati. Essi sono:

1) Sistemare insieme i moduli può costituire un grosso problema, in particolare se sono diverse le persone che scrivono i moduli.
2) I moduli richiedono una documentazione molto accurata, poichè essi possono riguardare altre parti del programma, come le strutture dati usate da tutti i moduli.
3) Il collaudo ed il debugging dei moduli separatamente risultano difficili, poiché i dati impiegati dal modulo collaudato possono essere prodotti da altri moduli come pure i risultati possono essere usati da altri moduli. Ci si può trovare nella necessità di scrivere programmi speciali (detti «driver») proprio per produrre dati campione e per provare i programmi. Questi 'driver' richiedono uno sforzo di programmazione ulteriore che nulla aggiunge però al sistema.
4) I programmi possono presentarsi difficili da modularizzare. Se si modularizza male il programma, l'integrazione risulterà veramente difficoltosa, poichè quasi tutti gli errori e le modifiche interesseranno parecchi moduli.
5) I programmi modulari spesso richiedono tempi e memoria in più, poichè i singoli moduli separati possono ripetere delle funzioni.

Dunque, mentre la programmazione modulare è certamente un miglioramento rispetto al tentativo di scrivere l'intero programma dal nulla, ci sono anche degli svantaggi.

Importanti considerazioni comprendono il restringere l'ammontare totale di informazioni scambiate tra i moduli, il limitare quelle decisioni di progetto che sono soggette a modificarsi per un singolo modulo e il restringere l'accesso da un modulo ad un altro.[3]

PRINCIPI DI MODULARIZZAZIONE

Un problema evidente è che non esistono metodi provati, sistematici per modularizzare i programmi. Si menzioneranno i seguenti principi[4]:

1) I moduli che fanno riferimento a dati comuni dovrebbero far parte dello stesso modulo globale.
2) Due moduli tali che il primo usa o dipende dal secondo, ma non viceversa, dovrebbero essere separati.
3) Un modulo che serve a più di un altro modulo dovrebbe appartenere ad un modulo complessivo differenti dagli altri.
4) Due moduli di cui il primo serve a molti altri moduli e il secondo solo ad alcuni dovrebbero essere separati.
5) Due moduli le cui frequenze di utilizzo sono diverse in modo significativo dovrebbero appartenere a moduli diversi.
6) La struttura o l'organizzazione di dati in relazione tra loro dovrebbe essere contenuta in un singolo modulo.

Se risultasse veramente difficile modularizzare un programma, ciò può essere un sicuro indice della scarsa definizione del problema, ed è richiesta una ridefinizione. Troppi casi speciali, che richiedano una gestione particolare, o l'uso di un enorme numero di variabili, che richiedano speciali procedure, sono problemi che possono essere efficientemente trattati, ridefinendo i compiti a mano.

ESEMPI

Risposta ad un Interruttore

Questo semplice programma può essere diviso in due moduli:

MODULARIZZAZIONE DEL SISTEMA CON INTERRUTTORE E LAMPADA

Il Modulo 1 aspetta che l'interruttore sia acceso e accende la lampada come risposta.

Il Modulo 2 produce il ritardo di un secondo.

Il Modulo 1 è verosimilmente specifico del sistema, in quanto dipenderà da come l'interruttore e la sorgente vengono connessi. Il Modulo 2 avrà un'utilità generale, poiché molti compiti richiedono dei ritardi. Chiaramente, dovrebbe essere vantaggioso avere a disposizione un modulo di ritardo standard che possa fornire dei ritardi di lunghezza variabile. Il modulo richiederà un'accurata documentazione di modo che si sappia come determinare la lunghezza del ritardo, come chiamare il modulo, e quali registri e locazioni di memoria vengono interessati dal modulo.

Una versione generale del Modulo 1 sarebbe molto meno utile, poiché dovrebbe avere a che fare con differenti tipi e connessioni di interruttori e lampade.

Si potrebbe probabilmente trovare più semplice scrivere un modulo per una particolare configurazione di interruttori e lampade piuttosto che tentare di servirsi di una 'routine' standard. Si noti la differenza tra questa situazione ed il Modulo 2.

Il Caricatore di Memoria basato su Interruttori

MODULARIZZAZIONE DEL CARICATORE DI MEMORIA BASATO SU INTERRUTTORI

Il caricatore di memoria con interruttori è difficile da modularizzare, dal momento che tutti i compiti di programmazione dipendono dalla configurazione hardware ed i compiti sono così semplici che a fatica sembra che valga la pena di ricorrere a dei moduli. Il diagramma di flusso in Figura 13-9 suggerisce che un modulo potrebbe essere quello che attende una pressione di uno dei tre pulsanti da parte dell'operatore.

Altri moduli potrebbero essere:

- Un modulo di ritardo che fornisca il ritardo richiesto per eliminare i rimbalzi degli interruttori.
- Un modulo di display e interruttori che legga il dato dagli interruttori e lo invii ai display.
- Un modulo di lamp-test.

Moduli fortemente dipendenti dal sistema, quali gli ultimi due è improbabile che abbiano utilità generale. In questo esempio la programmazione modulare non offre molti vantaggi.

Il Terminale di Verifica

MODULARIZZAZIONE DEL TERMINALE DI VERIFICA

Il terminale di verifica, d'altro canto, si presta molto bene ad una programmazione modulare. L'intero sistema **può essere facilmente diviso in tre moduli principali:**

- **Modulo display e tastiera.**
- **Modulo trasmissione dati.**
- **Modulo ricezione dati.**

Un modulo generale di display e tastiera potrebbe gestire molti sistemi basati su tastiera e display. I sottomoduli dovrebbero eseguire compiti come:

- Riconoscere una nuova entrata da tastiera e andare a prendere il dato.
- Azzerare l'array in risposta al tasto 'Azzera'.
- Introdurre digit nella memoria.
- Cercare il carattere trailer (terminatore) o il tasto 'Invia'.
- Visualizzare i digit.

Sebbene le interpretazioni dei tasti e il numero dei digit varieranno, l'ingresso elementare, l'immagazzinamento dei dati, ed i procedimenti di visualizzazione dei dati saranno identici per un gran numero di programmi. Tasti di funzione come 'Azzera' dovrebbero essere standard. Chiaramente, **il progettista deve tener presente quali sono i moduli per altre applicazioni e porvi particolare attenzione.**

Il modulo di trasmissione dati potrebbe inoltre essere ripartito in sotto-moduli quali:

1) Aggiungere il carattere di header.
2) Trasmettere caratteri quando la linea d'uscita sia in grado di gestirli.
3) Generare tempi di ritardo tra due bit o due caratteri.
4) Aggiungere il carattere di trailer.
5) Controllare omissioni in trasmissione; cioè, non riconoscimento o incapacità a trasmettere senza errori.

Il modulo di ricezione dati potrebbe includere sotto-moduli quali:

1) Cercare il carattere di header.
2) Controllare l'indirizzo di destinazione del messaggio nei confronti dell'indirizzo di terminale.
3) Immagazzinare ed interpretare il messaggio.
4) Cercare il carattere di trailer.
5) Generare ritardi nei bit o nei caratteri.

PRINCIPIO DI CELARE LE INFORMAZIONI

Si noti che qui è importante implementare in un unico modulo ogni decisione (come la velocità dei bit, il formato del messaggio, o la procedura del controllo di errore). Una variazione in una di queste decisioni richiederà poi che si cambi solo quel singolo modulo. Gli altri moduli dovrebbero essere scritti in modo che siano completamente ignari dei valori scelti o dei metodi usati per l'implementazione del modulo. **Qui un importante concetto è il «principio di tener nascoste le informazioni»[5], in base al quale i moduli si scambiano soltanto le informazioni assolutamente essenziali per svolgere un dato compito. Ulteriori informazioni sono nascoste dentro un singolo modulo.**

Un uso importante di questo principio si ha nella gestione dell'errore. Ogni volta che un modulo rivela un errore letale, non inizieranno le procedure di ricupero. Invece, lo stato di errore verrà rinviato al modulo chiamante e permetterà a quest'ultimo di prendere la decisione su come riprendersi dall'errore. La spiegazione di ciò sta nel fatto che la procedura di livello inferiore non ha spesso sufficienti informazioni per decidere adeguatamente quali procedure di ripristino siano necessarie. Per esempio, si supponga di avere un modulo che accetti ingressi numerici da un utilizzatore. Questo modulo termina normalmente quando l'utilizzatore introduce una stringa di digit numerici chiusa da un ritorno carrello. L'entrata di caratteri non numerici forza il modulo a finire immediatamente in modo anormale. Dal momento che il modulo non conosce in quale contesto venga usato (cioè, è parte di un assemblatore, di un editor interattivo, o di un sistema di gestione di file?), non è in grado di prendere una valida decisione su come agire quando incontra un carattere non valido. Se è stato progettato nel modulo un singolo metodo di ripristino da errore, tale modulo perderà la sua generalità e diventerà specifico per quelle situazioni che impiegano tale tecnica di ripristino da errore.

ANALISI DELLA PROGRAMMAZIONE MODULARE

REGOLE PER LA PROGRAMMAZIONE MODULARE

La programmazione modulare può risultare veramente vantaggiosa se ci si conforma alle seguenti regole:

1) **Impiegare moduli da 20 a 50 linee.** Moduli più corti sono usualmente uno spreco di tempo, mentre moduli più lunghi sono raramente di utilità generale e possono risultare difficili da integrare.
2) **Tentare di costruire moduli ragionevolmente generali.** Differenziare tra caratteristiche comuni come il codice ASCII o i formati di trasmissione asincrona, che saranno identici in molte applicazioni o identificazioni di tasti, e come il numero di display o di caratteri in un messaggio, che sono probabilmente unici per una particolare applicazione. Rendere semplice la variazione dei parametri più recenti. Le variazioni maggiori, come i codici di caratteri differenti, dovrebbero essere gestite da moduli separati.
3) **Prendere tempo supplementare nei moduli** come ritardi, gestori di display, di tastiera ecc., **che sarà utile in altri progetti o in molti punti diversi del programma attuale.**
4) **Tentare di mantenere i moduli il più possibile distinti e logicamente separati.** Restringere il flusso di informazione tra i moduli ed implementare ogni decisione di progetto in un singolo modulo.
5) **Non cercare di modularizzare dei compiti semplici,** mentre riscrivere l'intero compito può presentarsi più facile che assemblare o modificare il modulo.

PROGRAMMAZIONE STRUTTURATA

Come mantenere distinti dei moduli ed impedire loro di interagire? Come scrivere un programma che abbia una sequenza di operazioni così evidenti da permettere di isolare e di correggere gli errori? Una risposta sarebbe quella di ricorrere al metodo conosciuto come «programmazione struttura», per mezzo del quale ogni parte del programma è costituita da elementi di un limitato insieme di strutture e ogni struttura ha una singola entrata ed una singola uscita.

La Figura 13-14 mostra un diagramma di flusso di un programma non strutturato. Se avviene un errore nel modulo B, si hanno cinque possibili sorgenti di tale errore. Non solo si deve controllare ciascuna sequenza, ma ci si deve anche assicurare che ogni variazione, fatta per correggere l'errore, non interessi altre sequenze. Il risultato usuale è che il debugging diventa come combattere una piovra. Ogni volta che si pensa di avere la situazione sotto controllo, c'è un tentacolo libero in qualche posto.

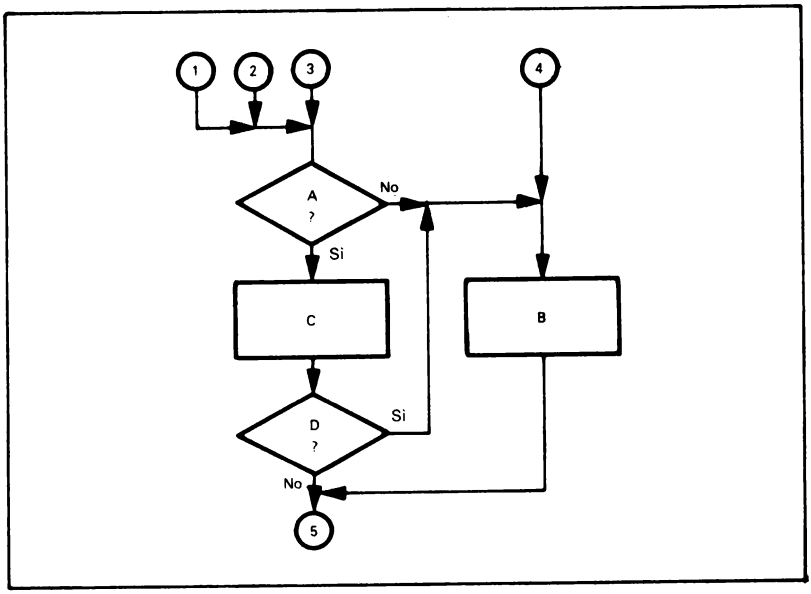


Figura 13-14. Diagramma di Flusso di un Programma non Strutturato

STRUTTURE DI BASE DELLA PROGRAMMAZIONE STRUTTURATA

La soluzione consiste nello stabilire una sequenza chiara di operazioni tale da permettere di isolare gli errori. Una siffatta sequenza impiega moduli ad entrata ed uscita singoli. **I moduli di base necessari sono:**

1) **Una sequenza ordinaria;** cioè una struttura lineare nella quale gli statement o le strutture sono eseguite consecutivamente. Nella sequenza:

S1
S2
S3

il computer esegue prima S1, poi S2 e infine S3. S1, S2 e S3 possono essere singole istruzioni o interi programmi.

2) **Una struttura condizionale.**
La più comune è «if C then S1 else S2» (Se C allora S1 altrimenti S2), dove C è una condizione e S1 e S2 sono istruzioni o sequenze di istruzioni. Il computer esegue S1 se C è vera e S2 se C è falsa. La Figura 13-15 mostra la logica di questa struttura. Si noti che la struttura ha una singola entrata ed una singola uscita; non c'è modo di entrare in S1 o S2 o di uscirne se non attraverso la struttura.

3) **Una struttura a ciclo.**
La struttura comune a ciclo è «while C do S» (finché C fai S), dove C è una condizione ed S è un'istruzione o una sequenza di istruzioni: Il computer controlla C ed esegue S se C è vera. Questa struttura (vedere Figura 13-16) inoltre ha una singola entrata ed una singola uscita. Si noti che il computer non eseguirà S per niente se C è originariamente falsa, poiché il valore di C è controllato prima di eseguire S.

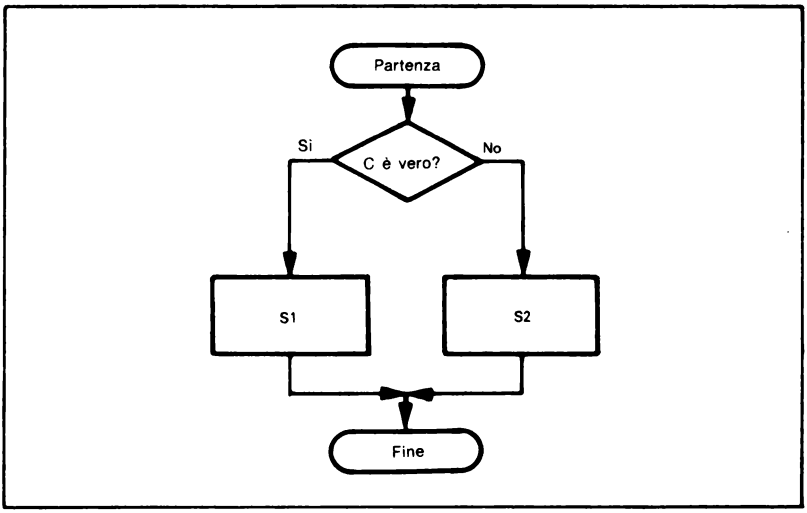


Figura 13-15. Diagramma di Flusso di una Struttura Se-Allora-Altrimenti

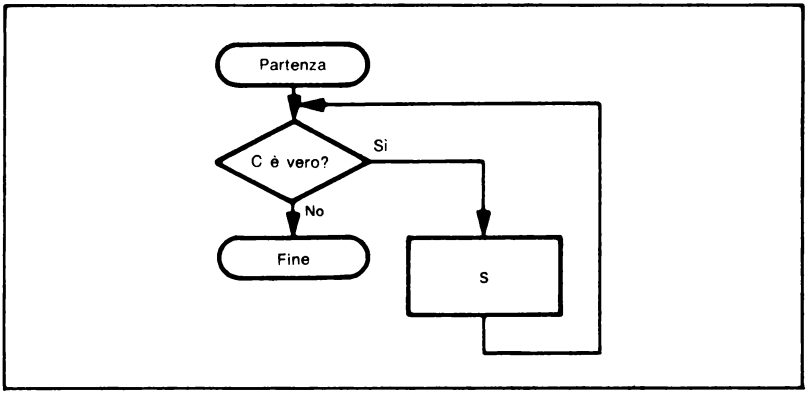


Figura 13-16. Diagramma di Flusso della Struttura Esegue-Finchè

Nella maggior parte dei linguaggi di programmazione strutturata, è prevista la costruzione di un ciclo alternativo. Questa costruzione è conosciuta come la clausola do-until (fai-fino a). La sua struttura è «do S until C» (fai S fino a C), dove C è una condizione ed S è un'istruzione o sequenza di istruzioni. Essa è simile alla costruzione do-while eccetto che la condizione C di confronto del ciclo è eseguita alla fine del ciclo. Ciò produce l'effetto di garantire sempre l'esecuzione del ciclo almeno una volta. Ciò viene illustrato dal Diagramma di Flusso in Figura 13-17. Il ciclo comune di DO o controllato da indice può essere implementato come caso speciale di entrambe queste due costruzioni di ciclo basilari.

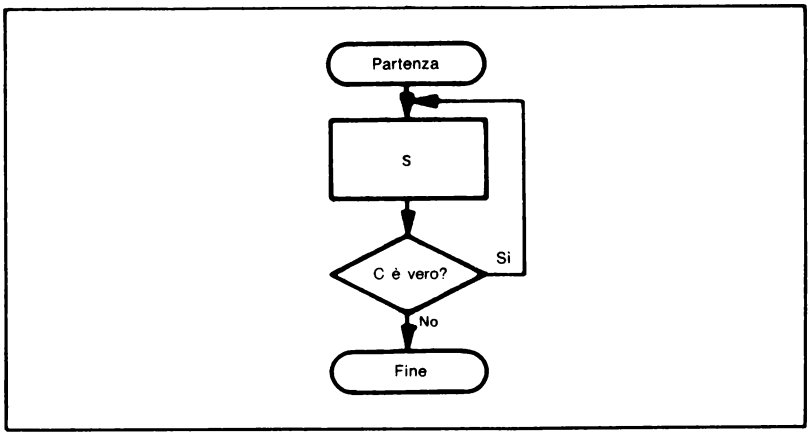


Figura 13-17. Diagramma di Flusso di una Struttura Do-Until.

4) **Una struttura casuale.**
Sebbene non sia una struttura primitiva come quella sequenziale, if-then-else, e do-while, la struttura casuale è usata tanto comunemente che la si include qui come un'aggiunta alle descrizioni di strutture di base. La struttura casuale è «case I of S0, S1,... Sn» (caso I di S0, S1,... Sn) dove I è un indice e S0, S1... Sn sono statement o sequenze di statement. Se I è uguale a zero allora viene eseguito lo statement S0; se I è uguale a 1 allora si esegue lo statement S1, ecc. Soltanto uno degli n-statement viene eseguito. Dopo la sua esecuzione, il controllo passa al prossimo statement sequenziale che segue nel gruppo di statement casuale. Se I è maggiore di n (cioè il numero di statement nello statement casuale), allora nessuno degli statement nello statement casuale è eseguito e il controllo è passato direttamente al successivo statement sequenziale seguente lo statement casuale. Questo è illustrato dal diagramma di flusso in Figura 13-18.

Si notino le seguenti caratteristiche della programmazione strutturata:

1) **Soltanto le tre strutture di base, e possibilmente un piccolo numero di strutture ausiliarie, sono permesse.**
2) **Le strutture possono essere inserite l'una nell'altra ad un qualsiasi livello di complessità cosicché un programma può, a sua volta, contenere una delle strutture.**
3) **Ciascuna struttura ha una singola entrata ed una singola uscita.**

Alcuni esempi della struttura condizionale illustrata in Figura 13-15 sono:

ESEMPI DI STRUTTURE

1) S2 inserito:
if X $\geq$ 0 then NPOS = NPOS + 1
else NNEG = NNEG + 1

Sia S1 che S2 sono singoli statement.

2) S2 omesso:
if X $\neq$ 0 then Y = 1/X

Qui non c'è alcuna azione se C (X$\neq$ 0) è falsa. S2 ed «else» possono essere omessi in questo caso.

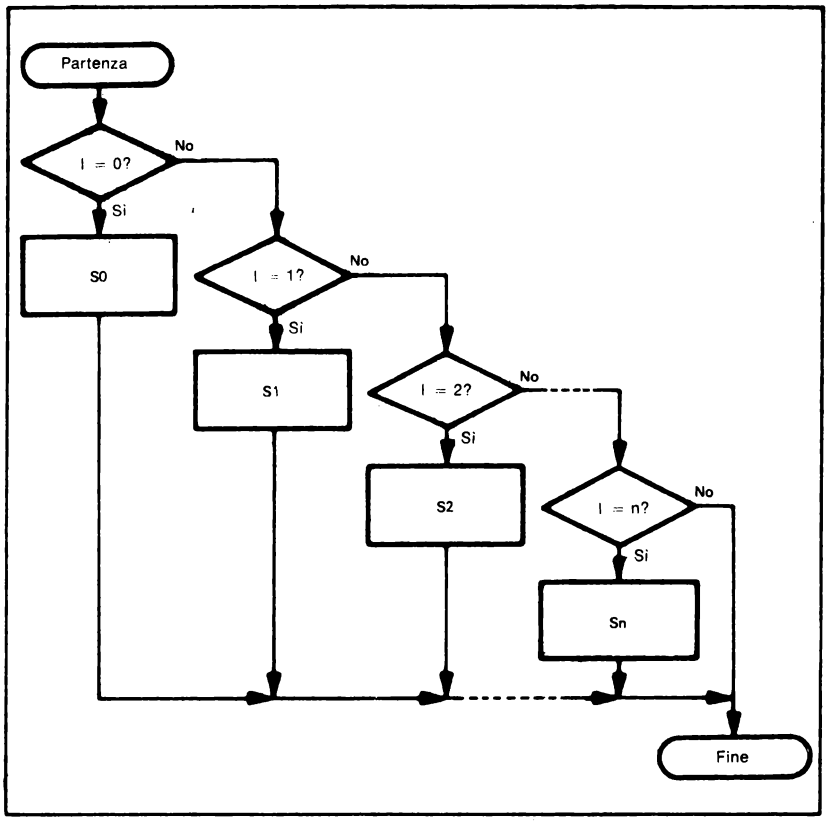


Figura 13-18. Diagramma di Flusso della Struttura Casuale.

Alcuni esempi della struttura a ciclo illustrata in Figura 13-16 sono:

1) Formare la somma di interi da 1 a N.

```
I = 0
SUM = 0
do while I < N
   I = I + 1
   SUM = SUM + I
end
```

Il computer esegue il ciclo finché I < N. Se N = 0, il programma compreso nel «do-while» non è eseguito affatto.

2) Contare i caratteri in un array SENTENCE fino a trovare un punto ASCII.

```
NCHAR = 0
do while SENTENCE (NCHAR) ≠ PERIOD
   NCHAR = NCHAR + 1
end
```

Il computer esegue il ciclo finché il carattere in SENTENCE non è un periodo ASCII. Il calcolo è zero se il primo carattere è un periodo.

I vantaggi della programmazione strutturata sono:

VANTAGGI DELLA PROGRAMMAZIONE STRUTTURATA

1) La sequenza delle operazioni è semplice da tracciare. Questo permette di eseguire facilmente il collaudo ed il debugging.
2) Il numero di strutture è limitato e la terminologia è standardizzata.
3) Le strutture possono essere facilmente costruite nei moduli.
4) I teorici hanno dimostrato che l'insieme di strutture è completo: cioè tutti i programmi possono essere scritti nei termini delle tre strutture.
5) La versione strutturata di un programma è in parte auto-documentante e abbastanza facile da leggere.
6) I programmi strutturati sono facili da descrivere con bozze di programma.
7) La programmazione strutturata aumenta, come è stato dimostrato in pratica, la produttività del programmatore.

La programmazione strutturata comporta principalmente una maggiore disciplina da parte del programmatore di quanta non ne richiedesse la programmazione modulare. Il risultato porta a programmi più sistematici e meglio organizzati.

Gli svantaggi della programmazione strutturata sono:

SVANTAGGI DELLA PROGRAMMAZIONE STRUTTURATA

1) Solo alcuni linguaggi ad alto livello (ad esempio PL/M, PASCAL) accetteranno direttamente le strutture. Il programmatore allora deve affrontare uno stadio in più di traduzione per convertire le strutture nel codice di linguaggio assembly. La versione strutturata del programma, comunque, è spesso utile come documentazione.
2) I programmi strutturati spesso sono di esecuzione più lenta e richiedono maggiore memoria rispetto a quella usata da programmi non strutturati.
3) Limitandosi alle quattro forme base di strutture si rendono certi compiti veramente difficili da realizzare. La completezza delle strutture significa soltanto che tutti i programmi possono essere implementati con esse; non comporta però che un dato programma sia implementato efficientemente o convenientemente.
4) Le strutture standard sono spesso abbastanza disorientanti, per esempio le strutture «if-then-else» incluse una nell'altra possono risultare molto difficili da leggere, poiché può mancare una chiara indicazione di dove sia la loro fine all'interno. Una serie di cicli «do-while» inseriti l'uno nell'altro può anch'essa essere di difficile lettura.
5) I programmi strutturati prendono in considerazione soltanto la sequenza di operazioni di un programma, non il flusso dei dati. Perciò, le strutture gestiscono i dati in modo inadeguato.
6) Pochi programmatori sono abituati alla programmazione strutturata. Molti trovano le strutture standard scomode e restrittive.

Non si invoglia né si scoraggia l'uso della programmazione strutturata. È un modo di stendere un programma in maniera sistematica. In generale, i linguaggi della programmazione strutturata, ad alto livello, sono utili maggiormente nelle seguenti situazioni:

QUANDO USARE LINGUAGGI AD ALTO LIVELLO PER PROGRAMMAZIONE STRUTTURATA

- Programmi più estesi, forse eccedenti 1000 istruzioni.
- Applicazioni in cui l'uso della memoria non è critico.
- Applicazioni di basso-volume dove i costi dello sviluppo software, particolarmente il collaudo ed il debugging sono fattori importanti.
- Applicazioni che comportano manipolazione di stringhe, controllo di processo, o altri algoritmi piuttosto che manipolazioni di bit.

In futuro, ci si attende una diminuzione del costo della memoria, un aumento della dimensione media dei programmi di microprocessore ed un aumento del costo dello sviluppo software. Perciò, metodi come l'uso dei linguaggi di programmazione strutturata ad alto livello, che abbassano i costi dello sviluppo software per programmi più estesi ma con uso di maggiore memoria, diventeranno di maggior valore.

Proprio perchè i concetti della programmazione strutturata sono normalmente espressi in linguaggi ad alto livello non significa che la programmazione strutturata non sia applicabile alla programmazione in linguaggio assembly. Al contrario, il programmatore in linguaggio assembly, con la totale libertà di espressione che la programmazione di livello assembly permette, ha bisogno dei concetti di strutturazione presenti nella programmazione strutturata. Creare moduli con delle singole entrate e delle singole uscite, usare semplici strutture di controllo e mantenere il minimo grado di complessità in ciascun modulo, tutto contribuisce ad una più efficiente codifica in linguaggio assembly.

ESEMPI

Risposta ad un Interruttore

PROGRAMMAZIONE STRUTTURATA NEL SISTEMA DI INTERRUTTORE E LAMPADA

La versione strutturata di questo esempio è:

```
SWITCH = OFF
do while SWITCH = OFF
  READ SWITCH
end
LIGHT = ON
DELAY 1
LIGHT = OFF
```

ON e OFF devono avere le adeguate definizioni per l'interruttore e per la luce. Si assume che DELAY è un modulo che produce un dato ritardo con i suoi parametri in secondi.

Uno statement in un programma strutturato può essere effettivamente un sottoprogramma. Comunque, volendosi attenere alle regole della programmazione strutturata, il sottoprogramma non può avere altre uscite all'infuori di quella che rimanda il controllo al programma principale.

Poiché «fai-finché» controlla la condizione prima di eseguire il ciclo, si posiziona la variabile SWITCH nello stato OFF prima di iniziare. Comunque nel caso si debba inizializzare SWITCH e si possano combinare le procedure di lettura e di controllo, probabilmente si dovrà ricorrere a qualcosa in più di memoria rispetto al programma non strutturato.

Il Caricatore di Memoria basato su Interruttori

PROGRAMMAZIONE STRUTTURATA PER IL CARICATORE DI MEMORIA BASATO SU INTERRUTTORI

Il caricatore di memoria basato su interruttori è un problema di programmazione strutturata più complesso. Si può implementare il diagramma di flusso della Figura 13-19 come segue (un • indica un commento):

```
•
• INIZIALIZZA LE VARIABILI
•
HIADDRESS = 0
LOADDRESS = 0
•
```

```
• QUSTO PROGRAMMA USA UNA STRUTTURA DO-WHILE SENZA CONDIZIONI
• (DENOMINATA SEMPLICEMENTE DO-FOREVER). PERCIÒ, IL SISTEMA ESEGUE
• CONTINUAMENTE IL PROGRAMMA CONTENUTO IN QUESTO CICLO DO-WHILE.
•
do forever
•
• PROVA SUL TASTO HIADDRESS; REALIZZA L'ELABORAZIONE RICHIESTA
• SE IL TASTO È ON.
•
  if HIADDRBUTTON = 1 then
   begin
    HIADDRESS = SWITCHES
    LIGHTS = SWITCHES
    do
      DELAY (DEBOUNCE TIME)
    until HIADDRBUTTON≠1
  end
•
• PROVA SUL TASTO LOADDRESS; REALIZZA L'ELABORAZIONE DELL'INDIRIZZO BASSO
• SE IL TASTO È ON.
•
  if LOADDRBUTTON = 1 then
   begin
    LOADDRESS = SWITCHES
    LIGHTS = SWITCHES
    do
      DELAY (DEBOUNCE TIME)
    until LOADDRBUTTON≠ 1
  end
•
• PROVA SUL TASTO DATI, E MEMORIZZAZIONE DEL DATO IN MEMORIA
• SE IL TASTO È ON.
•
  if DATABUTTON = 1 then
   begin
    DATA = SWITCHES
    LIGHTS = SWITCHES
    (HIADDRESS, LOADDRESS) = DATA
    do
      DELAY (DEBOUNCE TIME)
    until DATABUTTON≠ 1
  end
•
• L'ULTIMO END DI SOPRA FA TERMINARE IL
•          do forever LOOP
•
```

I programmi strutturati non sono facili da scrivere, ma possono far comprendere a fondo la logica complessiva del programma. Si può controllare la logica del programma strutturato trattando qualsiasi codice effettivo prima di scrivere.

Il Terminale di Verifica di Credito

PROGRAMMA STRUTTURATO PER IL TERMINALE DI VERIFICA DI CREDITO
ROUTINE STRUTTURATA DI TASTIERA

Guardiamo all'entrata da tastiera per il terminale di transazione. Si assumerà che il vettore di visualizzazione sia ENTRY, lo strobe di tastiera sia KEYSTROBE e il dato da tastiera sia KEYIN. **Il programma strutturato senza i tasti di funzione è:**

```
NKEYS = 10
•
• AZZERA L'ENTRATA PER PARTIRE
•
  do while NKEYS > 0
    NKEYS = NKEYS - 1
    ENTRY (NKEYS) = 0
  end
•
• VAI A PRENDERE UNA ENTRATA COMPLETA DA TASTIERA
•
  do while NKEYS < 10
    if KEYSTROBE = ACTIVE then
      begin
        KEYSTROBE = INACTIVE
        ENTRY (NKEYS) = KEYIN
        NKEYS = NKEYS + 1
      end
  end
```

Aggiungere il tasto SEND significa che il programma deve ignorare i digit in più dopo che si è verificata una entrata completa, e deve ignorare il tasto SEND fino a che esso non abbia un'entrata completa. Il programma strutturato è:

```
NKEYS = 10
•
• AZZERA L'ENTRATA PER PARTIRE
•
  do while NKEYS < 0
      NKEYS = NKEYS - 1
      ENTRY (NKEYS) = 0
  end
•
• ATTENDI UNA ENTRATA COMPLETA SEGUITA DAL TASTO SEND
•
  do while KEY ≠ SEND OR NKEYS ≠ 10
    if  KEYSTROBE = ACTIVE then
      begin
        KEYSTROBE = INACTIVE
        KEY = KEYIN
        if NKEYS ≠ 10 and KEY ≠ SEND then
        begin
            ENTRY (NKEYS) = KEY
          NKEYS = NKEYS + 1
        end
    end
end
```

Si notino le seguenti caratteristiche di questo programma strutturato:

1) Il secondo if-then è inserito nel primo, poiché i tasti sono introdotti solo dopo che uno strobe venga riconosciuto. Se il secondo if-then fosse allo stesso livello del primo, un singolo tasto potrebbe riempire l'entrata, in quanto il suo valore sarebbe introdotto nel vettore durante ciascuna iterazione del ciclo do-while.
2) KEY non ha bisogno di essere definitiva inizialmente, poiché NKEYS è messa a zero come parte dell'azzeramento dell'entrata.

Aggiungere il tasto CLEAR permette al programma di azzerare l'entrata in origine simulando la pressione di CLEAR; cioè, mettendo NKKEYS a 10 e KEY a CLEAR prima di partire. Il programma strutturato deve inoltre azzerare solo quei digit che sono stati precedentemente occupati. **Il nuovo programma strutturato è:**

```
•
• SIMULA UN AZZERAMENTO COMPLETO
•
NKEYS = 10
KEY = CLEAR
•
• ATTENDI UNA ENTRATA COMPLETA E IL TASTO SEND
•
do while KEY ≠ SEND O NKEYS ≠ 10
•
•  AZZERA TUTTA L'ENTRATA SE È STATO PREMUTO IL TASTO CLEAR
•
  if KEY = CLEAR then
   begin
     KEY = 0
     do while NKEYS > 0
     NKEYS = NKEYS - 1
     ENTRY (NKEYS) = 0
   end
 end
•
• PRENDI IL DIGIT SE L'ENTRATA È INCOMPLETA
•
  if KEYSTROBE = ACTIVE then
   begin
     KEYSTROBE = INACTIVE
     KEY = KEYIN
     if KEY < 10 and NKEYS ≠ 10 then
      begin
        ENTRY (NKEYS) = KEY
        NKEYS = NKEYS + 1
         end
     end
  end
```

Si noti che il programma opera il reset di KEY a zero dopo aver azzerato il vettore, cosicché l'operazione non viene ripetuta.

ROUTINE STRUTTURATA DI RICEZIONE

Si può in maniera del tutto simile costruire un programma per la routine di ricezione. Un programma iniziale potrebbe cercare proprio i caratteri di header e di trailer. Si assumerà che RTSB sia l'indicatore che un carattere è pronto. **Il programma strutturato è:**

```
•
• AZZERA IL FLAG HEADER PER PARTIRE
•
HFLAG := 0
•
• ATTENDI UN HEADER E UN TRAILER
•
do while HFLAG = 0 oppure CHAR ≠ TRAILER
•
• PRENDI IL CARATTERE SE È PRONTO, CERCA UN HEADER
•
  if RSTB = ACTIVE then
   begin
     RSTB = INACTIVE
     CHAR = INPUT
     if CHAR = HEADER then HFLAG = 1
   end
```

Ora si può aggiungere la sezione che controlla l'indirizzo di messaggio rispetto ai tre digit dell'INDIRIZZO DI TERMINALE (TERMADDR). Se uno dei digit corrispondenti risultasse diverso, il flag di RISCONTRO D'INDIRIZZO (ADDRMATCH) viene messo a 1.

```
•
• AZZERA IL FLAG DI HEADER, IL FLAG DI RISCONTRO DI INDIRIZZO, IL
  CONTATORE DI INDIRIZZO PER PARTIRE
•
HFLAG = 0
ADDRMATCH = 0
ADDRTR = 0
•
• ATTENDI HEADER, INDIRIZZO DI DESTINAZIONE, E TRAILER
•
do while HFLAG = 0 oppure CHAR ≠ TRAILER OPPURE ADDRCTR ≠ 3
•
• PRENDI IL CARATTERE SE È PRONTO
•
  if RSTB = ACTIVE then
   begin
     RSTB = INACTIVE
     CHAR = INPUT
   end
•
• CONTROLLO L'INDIRIZZO TERMINALE E L'HEADER
•
  if HFLAG = 1 and ADDRCTR ≠ 3 then
   begin
     ADDRMATCH = 1
     ADDRCTR = ADDRCTR + 1
   end
  if CHAR = HEADER then HFLAG = 1
end
```

Il programma deve ora aspettare un header, un codice di identificazione a tre digit ed un trailer. Si deve fare attenzione a cosa si verifica durante l'iterazione quando il programma trova l'header, e a cosa si verifica se un carattere erroneo di codice di identificazione è lo stesso del trailer.

Una ulteriore aggiunta può immagazzinare il messaggio in MESSG. NMESS è il numero di caratteri nel messaggio; se non c'è zero alla fine, il programma ritiene che il terminale abbia ricevuto un messaggio valido. Non si è tentato di minimizzare le espressioni logiche in questo programma.

```
•
• AZZERA I FLAG, I CONTATORI PER PARTIRE
•
HFLAG = 0
ADDRMATCH = 0
ADDRCTR = 0
NMESS = 0
•
• ATTENDI HEADER, INDIRIZZO DI DESTINAZIONE E TRAILER
•
do while HFLAG = 0 oppure CHAR ≠ TRAILER oppure ADDRCTR ≠ 3
•
• PRENDI IL CARATTERE SE È PRONTO
•
  if RSTB = ACTIVE then
   begin
     RSTB = INACTIVE
     CHAR = INPUT
   end
•
• LEGGI IL MESSAGGIO SE INDIRIZZO DI DESTINAZIONE = INDIRIZZO DI TERMINALE
•
  if HFLAG = 1 and ADDRCTR = 3 then
   if ADDRMATCH = 0 and CHAR ≠ TRAILER then
   begin
     MESSG(NMESS) = CHAR
     NMESS = NMESS + 1
   end
•
• CONTROLLO SULL'INDIRIZZO TERMINALE
•
  if HFLAG = 1 and ADDRCTR ≠ 3 then
   if CHAR ≠ TERMADDR(ADDRCTR) then
   begin
     ADDRMATCH = 1
     ADDRCTR = ADDRCTR + 1
   end
•
• RICERCA HEADER
•
  if CHAR = HEADER then HFLAG = 1
end
```

Il programma controlla il codice di identificazione solo se trova un header durante una precedente iterazione. Esso accetta il messaggio solo se ha precedentemente trovato un header ed un completo riscontro dell'indirizzo di destinazione. Il programma deve lavorare correttamente durante le iterazioni quando trova l'header, il trailer e l'ultimo digit dell'indirizzo di destinazione. Non deve tentare di riscontrare l'header con l'indirizzo di terminale né piazzare il trailer o il digit finale dell'indirizzo di destinazione nel messaggio. **Si potrebbe tentare di aggiungere il resto della logica nel diagramma di flusso (Figura 13-13) al programma strutturato. Si noti che l'ordine delle operazioni spesso è critico. Si deve essere sicuri che il programma non porti a termine una fase e inizi la successiva durante la stessa iterazione.**

ANALISI DELLA PROGRAMMAZIONE STRUTTURATA

La programmazione strutturata porta ad una disciplina nella stesura di un programma. Obbliga a limitare i tipi di strutture da usare e la sequenza di operazioni. Fornisce strutture a singole entrate, a singole uscite, che si possono controllare per l'esattezza logica. La programmazione strutturata spesso rende il progettista conscio delle incompatibilità o delle possibili combinazioni degli ingressi. La programmazione strutturata non è una panacea, ma mette ordine in un processo che può essere caotico. La programmazione strutturata potrebbe inoltre essere d'aiuto nel debugging, nel collaudo e nella documentazione.

La programmazione strutturata non è semplice. Il programmatore non deve soltanto definire il problema in maniera adeguata, ma deve anche lavorare con la logica attentamente. Ciò è tedioso e difficile, ma ne deriva un programma funzionante e scritto chiaramente.

TERMINATORI E STRUTTURE

Le strutture particolari che sono state presentate non sono ideali e sono spesso difficoltose. In aggiunta, può risultare difficile individuare dove una struttura termini ed un'altra inizi, particolarmente se sono inserite l'una nell'altra. I teorici sono in grado di fornire per il futuro strutture migliori, o i progettisti possono desiderare di aggiungerne qualcuna delle proprie. Si rendono necessari alcuni tipi di terminatori per ciascuna struttura, dal momento che creare delle spaziature non sempre rende chiara la situazione. «End» è un terminatore logico per il ciclo «do-while». Non esiste un evidente terminatore, comunque, per l'istruzione «if-then-else»: alcuni teorici hanno suggerito «endif» o «fi» «if» all'inverso, ma questi risultano entrambi inopportuni e vanno a scapito della leggibilità del programma.

REGOLE PER LA PROGRAMMAZIONE STRUTTURATA

Si suggeriscono le seguenti regole da applicare alla programmazione strutturata:

1) **Iniziare con lo scrivere un diagramma di flusso di base** che aiuti a definire la logica del programma.
2) **Iniziare con le costruzioni «sequenziali», «if-then-else» e «do-while».** Esse sono conosciute come un insieme completo, cioè, qualsiasi programma può essere scritto con queste strutture.
3) **Distanziare ciascun livello** con pochi spazi dal livello precedente, di modo da riconoscere quali istruzioni ne fanno parte.
4) **Usare terminatori per ciascuna struttura;** ad esempio «end» per il «do-while» ed «endif» o «fi» per il «if-then-else». I terminatori più che la spaziatura dovrebbero rendere il programma ragionevolmente chiaro.
5) **Dare risalto alla semplicità e alla leggibilità.** Lasciare molti spazi, usare nomi significativi, e fare le espressioni il più chiare possibili. Non tentare di minimizzare la logica a prezzo della chiarezza.
6) **Commentare il programma** in modo organico.
7) **Controllare la logica.** Provare tutti i casi estremi o le condizioni speciali e alcuni casi campione. Qualsiasi errore di logica trovato a questo livello non tormenterà in seguito.

PROGETTAZIONE TOP-DOWN

PROGETTAZIONE BOTTOM-UP

Il problema rimanente è come controllare ed integrare moduli o strutture. Certamente si vuole dividere un compito vasto in dei sottocompiti. Ma come controllare i sotto-compiti isolatamente e metterli insieme? La procedura standard, detta «progettazione botton-up», richiede lavoro in più nel collaudo e nel debugging e lascia l'intero compito dell'integrazione alla fine. Ciò di cui si ha bisogno è un metodo che permetta un collaudo ed un debugging nella realizzazione del programma reale e modularizzi l'integrazione del sistema.

METODI DI PROGETTAZIONE TOP-DOWN

STUB

Questo metodo è la «progettazione top-down». Qui si inizia scrivendo il programma supervisore totale. Si rimpiazzano i sottoprogrammi non definiti con degli «stub» (tronconi) di programma, programmi temporanei che possono anche memorizzare l'entrata, fornire la risposta ad un selezionato problema di collaudo, o non fare niente. Allora si esamina il programma supervisore per vedere la correttezza della sua logica.

SVILUPPO DEGLI STUB

Si procede sviluppando gli stub. Ciascuno stub conterrà spesso dei sotto-compiti, che temporaneamente verranno rappresentati come stub. Questo processo di sviluppo, di debugging e di collaudo continua fino a che tutti gli stub non siano stati sostituiti da programmi funzionanti.

VANTAGGI DELLA PROGETTAZIONE TOP-DOWN

Si noti che il collaudo e l'integrazione intervengono in ogni livello, piuttosto che tutti alla fine. Non sono necessari né speciali driver né programmi di generazione dati. Si fornisce una idea chiara di dove esattamente si è nel progetto. La progettazione top-down ammette una programmazione modulare ed è compatibile anche con la programmazione strutturata.

Gli svantaggi di una progettazione top-down sono:

SVANTAGGI DI UNA PROGETTAZIONE TOP-DOWN

1) Il progetto complessivo può non ingranarsi bene con il sistema hardware.
2) Può non ottenere buoni vantaggi dal software esistente.
3) Gli stub possono essere difficili da scrivere, particolarmente se devono lavorare correttamente in diversi punti.
4) Il progetto top-down può non risultare utile in genere per i moduli.
5) Errori a livello top possono avere effetti catastrofici, mentre errori nel progetto top-down sono normalmente limitati ad un particolare modulo.

In molti progetti di programmazione, è stato mostrato che la progettazione top-down aumenta la produttività del programmatore. Comunque, la maggior parte di tutti questi progetti hanno impiegato delle stesure bottom-up laddove il metodo top-down avrebbe comportato un notevole lavoro in più.

La progettazione top-down è un mezzo utile che non dovrebbe essere seguita fino agli estremi. Essa fornisce la medesima disciplina di collaudo ed integrazione di sistema che la programmazione strutturata fornisce per la progettazione modulare. Il metodo comunque ha un campo di applicazione più generale, in quanto non presume l'uso di logica programmata. Tuttavia la progettazione top-down può non adattarsi alle implementazioni più efficienti.

ESEMPI

Risposta ad un Interruttore

PROGETTAZIONE TOP-DOWN DEL SISTEMA DI INTERRUTTORE E LAMPADA

Il primo esempio di programmazione strutturata dimostra in realtà il progetto top-down. Il programma era:

```
SWITCH = OFF
do while SWITCH = OFF
  READ SWITCH
end
LIGHT = ON
DELAY 1
LIGHT = OFF
```

Queste istruzioni sono in realtà degli stub, in quanto nessuna di esse è completamente definita. Per esempio, cosa significa READ SWITCH? Se l'interruttore fosse un bit della porta di ingresso SPORT, significherebbe realmente:

```
SWITCH = SPORT AND SMASK
```

dove SMASK ha un bit a '1' nell'appropriata posizione. La maschera può naturalmente essere implementata con una istruzione BIT TEST. In maniera simile, DELAY 1 significa in realtà (se il processore produce da sè il ritardo):

```
REG = COUNT
do while REG ≠ 0
  REG = REG - 1
end
```

COUNT è l'appropriato numero per generare un ritardo di un secondo. **La versione espansa del programma è:**

```
SWITCH = 0
do while SWITCH = 0
  SWITCH = SPORT AND MASK
end
LIGHT = ON
REG = COUNT
do while REG ≠ 0
  REG = REG - 1
end
LIGHT = NOT (LIGHT)
```

Certamente questo programma è più esplicito e potrebbe essere tradotto più facilmente in istruzioni o statement effettivi.

Il Caricatore di Memoria Basato su Interruttori

PROGETTAZIONE TOP-DOWN DI UN CARICATORE DI MEMORIA BASATO SU INTERRUTTORI

Questo esempio è più complesso del primo, cosicché si deve procedere sistematicamente. Pure qui, **il programma strutturato contiene degli stub.**
Per esempio, se il pulsante HIGH ADDRESS è un bit della porta di ingresso CPORT, «se HIADDRBUTTON = 1» significa in realtà:

1) Ingresso da CPORT.
2) Complemento.
3) AND logico con HAMASK.

dove HAMASK ha un «1» nel bit di posizione appropriata e «0» altrove. Ugualmente la condizione «if DATA BUTTON = 1» significa in realtà:

1) Ingresso da CPORT.
2) Complemento.
3) AND logico con DAMASK.

Perciò, gli stub iniziali potrebbero proprio assegnare dei valori ai pulsanti, ad esempio:

```
HIADDRBUTTON = 0
LOADDRBUTTON = 0
DATABUTTON = 0
```

Una esecuzione del programma supervisore mostrerebbe che esso porta la linea implicita «else» attraverso le strutture «if-then-else» e non legge mai gli interruttori. In maniera simile se lo stub fosse:

```
HIADDRBUTTON = 1
```

il programma supervisore si manterrebbe nel ciclo «do-while HIADDRBUTTON = 1» in attesa che il pulsante venga rilasciato. Queste semplici esecuzioni controllano la logica complessiva.

Ora si può espandere ciascuno stub ed esaminare se l'espansione produce un risultato ragionevolmente generale. Si noti come il debugging ed il collaudo procedano in maniera diretta e modulare. Si espanda lo stub HIADDRBUTTON = 1 con:

```
READ CPORT
HIADDRBUTTON = NOT (CPORT) AND HAMASK
```

Il programma aspetterà che il pulsante HIGH ADDRESS venga premuto. Il programma allora visualizzerà i valori degli interruttori sui LED. Questa esecuzione controlla la risposta appropriata al pulsante HIGH ADDRESS.

Si espande poi il modulo del pulsante LOW ADDRESS con:

```
READ CPORT
HIADDRBUTTON = NOT (CPORT) AND LAMASK
```

Con il pulsante LOW ADDRESS nella posizione premuta, il programma visualizzerà i valori degli interruttori sui LED. Questa esecuzione controlla la risposta adeguata al pulsante LOW ADDRESS. Similmente, si può espandere il modulo del pulsante DATA e controllare la risposta appropriata a quel pulsante. L'intero programma sarà stato allora collaudato.

Quando tutti gli stub sono stati espansi, gli stadi di codifica, debugging e collaudo verranno tutti completati. Naturalmente, si deve conoscere con esattezza quali sono i risultati che ciascuno stub dovrebbe produrre. Comunque, molti errori logici diventeranno evidenti a ciascun livello senza ulteriore espansione.

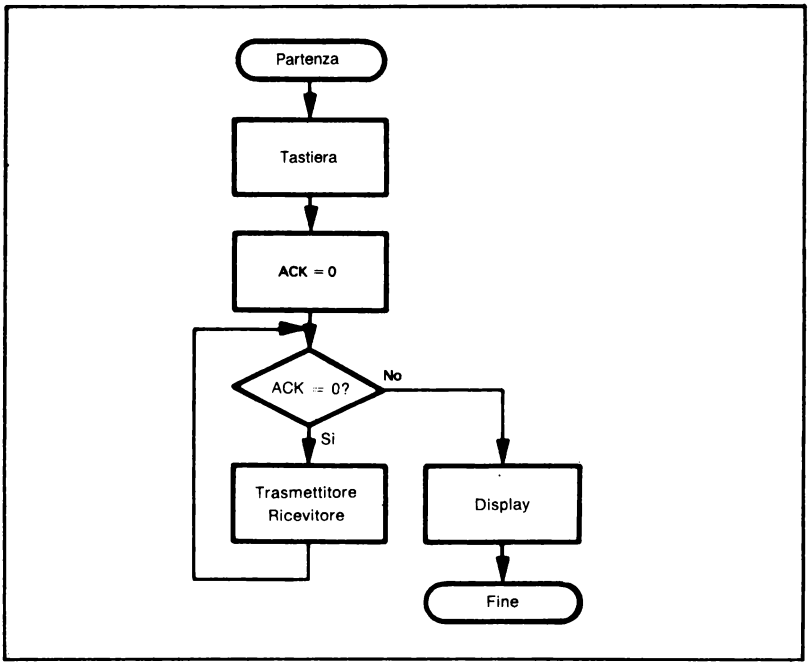


Figura 13-19. Diagramma di Flusso Iniziale per il Terminale di Operazione

Il Terminale di Transazione

PROGETTAZIONE TOP-DOWN DEL TERMINALE DI VERIFICA

Questo esempio naturalmente ha più livelli di dettaglio. Si potrebbe **iniziare con il seguente programma** (vedere la Figura 13-19 per un diagramma di flusso).

```
KEYBOARD
ACK = 0
do while ACK = 0
   TRANSMIT
   RECEIVE
end
DISPLAY
```

Qui KEYBOARD, TRANSMIT, RECEIVE e DISPLAY sono stub di programma che verranno espansi in seguito. KEYBOARD, ad esempio, potrebbe semplicemente depositare un numero verificato a 10 digit nell'appropriato accumulatore.

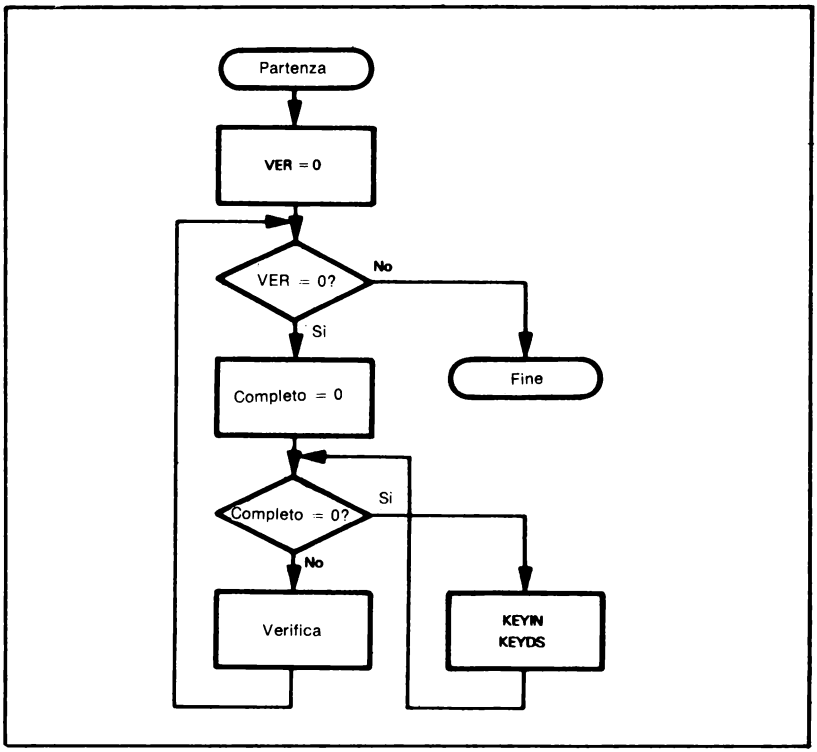


Figura 13-20. Diagramma di Flusso per una Routine KEYBOARD Ampliata

ESPANSIONE DELLA ROUTINE KEYBOARD

Il prossimo stadio di espansione potrebbe produrre il seguente programma per KEYBOARD (vedere la Figura 13-20):

```
VER = 0
do while VER = 0
   COMPLETE = 0
   do while COMPLETE = 0
      KEYIN
      KEYDS
   end
   VERIFY
end
```

Dove VER = 0 significa che un'entrata non è stata controllata; COMPLETE = 0 significa che l'entrata è incompleta. KEYIN e KEYDS sono rispettivamente le routine di ingresso da tastiera e di visualizzazione. VERYFY controlla l'entrata. Uno stub per KEYIN dovrebbe semplicemente mettere un'entrata casuale (da una tabella di numeri casuali o da un generatore) nell'accumulatore e porre COMPLETE a 1.

Si dovrebbe continuare una simile espansione, una simile messa a punto e collaudo di TRANSMIT, RECEIVE e DISPLAY. Si noti che si dovrebbe espandere ciascun programma ad un livello per volta di modo che non si esegua l'integrazione di un intero programma in una sola volta. Si deve usare giudizio nel definire i livelli. Passi troppo piccoli sprecano tempo, mentre passi troppo grandi rimandano ai problemi di integrazione del sistema che la progettazione top-down si supponeva avesse risolto.

ANALISI DELLA PROGETTAZIONE TOP-DOWN

Il progetto top-down implica disciplina negli stadi di collaudo ed integrazione della stesura del programma. Esso fornisce un metodo sistematico per espandere un diagramma di flusso o una definizione di problema al livello richiesto per scrivere effettivamente un programma. Insieme con la programmazione strutturata forma un insieme completo di tecniche di progetto.

Come la programmazione strutturata, il progetto top-down non è semplice. Il progettista deve aver definito il problema con accuratezza e deve procedere con un lavoro sistematico attraverso ciascun livello. Ancora una volta può sembrare che la metodologia sia tediosa, ma la ricompensa può essere fondamentale se si seguono le regole.

FORMATO PER LA PROGETTAZIONE TOP-DOWN

Si raccomanda il seguente approccio alla stesura top-down:

1) **Iniziare con un diagramma di flusso di base.**
2) **Rendere gli stub il più completi e separati possibile.**
3) **Definire con precisione tutte le possibili uscite di ciascuno stub** e selezionare un insieme di tests.
4) **Controllare ciascun livello con accuratezza** e sistematicamente.
5) **Usare** le strutture **della programmazione strutturata.**
6) **Espandere ciascuno stub ad un livello per volta.** Non tentare di fare troppo in un passo.
7) **Guardare con attenzione i compiti comuni e le strutture dati.**
8) **Collaudare ed eseguire il debugging dopo ogni espansione di uno stub.** Non tentare di fare un intero livello alla volta.
9) **Essere consapevoli del funzionamento dell'hardware. Non esitare nel fermarsi e nel fare un piccolo progetto botton-up laddove sembri necessario.**

ANALISI DELLA DEFINIZIONE DEL PROBLEMA E DEL PROGETTO DEL PROGRAMMA

Si potrebbe notare che è stato speso un intero capitolo senza menzionare un qualsiasi linguaggio specifico per microprocessore o assembly e senza scrivere una singola linea di codice effettivo. Con buone speranze, tuttavia, adesso si conoscono molte più informazioni sugli esempi di quelle che si avrebbero se fosse stato chiesto di scrivere i programmi all'inizio. Sebbene si pensa spesso che scrivere istruzioni sia una parte chiave dello sviluppo software, effettivamente è uno degli stadi più facili.

Una volta che si sono scritti alcuni programmi, la loro codifica diventerà semplice. Si apprenderà presto un insieme di istruzioni, si sarà in grado di distinguere quali istruzioni sono effettivamente utili, e si ricorderanno le sequenze comuni che compongono la parte più estesa della maggior parte dei programmi. Allora si scoprirà che molti degli altri stadi dello sviluppo software restano difficili e posseggono poche regole chiave.

Si è suggerito alcuni modi per rendere sistematici gli stadi importanti di prima. Nello stadio di definizione del problema si devono definire tutte le caratteristiche del sistema, i suoi ingressi, le sue uscite, le procedure, i limiti di tempo e di memoria, e la gestione errori. Si deve tenere in particolare considerazione come il sistema potrà interagire con il sistema più grande di cui fa parte e se quel sistema più grande include apparecchiature elettriche, meccaniche oppure operatori umani. Si deve iniziare da questo stadio per rendere il sistema facile da utilizzare e da mantenere.

Nello stadio di stesura del programma, diverse tecniche possono essere d'aiuto per specificare e documentare con sistematicità la logica del programma. La programmazione modulare obbliga a suddividere il programma globale in piccoli moduli distinti, mentre la stesura top-down è un metodo sistematico per integrarli e collaudarli. Naturalmente, nessuno può costringere a seguire tutte queste tecniche: esse sono, infatti, più che altro delle indicazioni direttive. Ma permettono un approccio unificato per la progettazione, pertanto dovrebbero essere considerate come una base sulla quale sviluppare il proprio approccio personale.

BIBLIOGRAFIA

1) Vedere, per esempio, V.P. Srini "Fault Diagnosis of Microprocessor Systems" Computer, Gennaio 1977, pp. 60-65. Per una descrizione della signature analysis, vedere G. Gordon e H. Nadig, "Hexadecimal Signatures Identify Trouble-spots in Microprocessor Systems" Electronics, 3 Marzo 1977, pp. 89-96. C'è anche una Application Note (#222) intitolata "A Designer's Guide to Signature Analysis" disponibile dalla Hewlett-Packard.

2. Per una breve discussione delle considerazioni sui fattori umani, vedere G. Morris, "Make Your Next Instrument Design Emphasize User Needs and Wants" EDN, 20 Ottobre 1978, pp. 100-105.

3. D.L. Parnas (vedere riferimenti seguenti) è stato una guida nell'area della programmazione modulare.

4. Raccolto da B.W. Unger (vedere riferimento seguente).

5. Formulato da D.L. Parnas.

I seguenti riferimenti forniscono informazioni aggiuntive sulla definizione del problema e la stesura del programma:

Chaplin, N., Flowcharts, Auerbach, Princeton, N. J., 1971.

Dahl, O. J., C. A. R. Hoave, and E. W. Dijkstra, Structured Programming, Academic Press, New York, N. Y., 1972.

Dalton, W. F., "Design Microcomputer Software like Other Systems – Systematically", Electronics, 19 Gennaio, 1978, pp. 97-101.

Dijkstra, E. W., A Discipline of Programming, Prentice-Hall, Englewood Cliffs, N. J., 1976.

Halstead, M. H., Elements of Software Science, American Elsevier, New York, 1977.

Hughes, J. K. and J. I. Michtom, A Structured Approach to Programming, Prentice-Hall, Englewood Cliffs, N. J., 1977.

Morgan, D. E. and D. J. Taylor, "A Survey of Methods for Achieving Reliable Software", Computer, Febbraio 1977, pp. 44-52.

Myers, W., "The Need for Software Engineering", Computer, Febbraio 1978, pp. 12-25.

Parnas, D. L., "On the Criteria to be Used in Decomposing Systems into Modules", Communications of the ACM, Dicembre 1972, pp. 1053-1058.

Parnas, D. L., "A Technique for the Specification of Software Modules with Examples", Communications of the ACM, Maggio 1973, pp. 330-336.

Schneider, V., "Prediction of Software Effort and Project Duration – Four New Formulas", SIGPLAN Notices, Giugno 1978, pp. 49-59.

Shneiderman, B. et al., "Experimental Investigations of the Utility of Detailed Flowcharts in Programming", Communications of the ACM, Giugno 1977, pp. 373-381.

Ulrickson, R. W., "Solve Software Modules Are the Building Blocks", Electronic Design, 1 Febbraio 1977, pp. 62-66.

Ulrickson, R. W., "Software Problems Step-by-Ste", Electronic Design, 18 Gennaio 1977, pp. 54-58.

Unger, B. W., "Programming Languages for Computer System Simulation", Simulation, Aprile 1978, pp. 101-110.

Wirth, N., Algorithms + Data Structures = Programs, Prentice-Hall, Englewood Cliffs, N. J., 1976.

Wirth, N., Systematic Programming; and Introduction, Prentice-Hall, Englewood Cliffs, N. J., 1973.

Yourdon, E. U., Techniques of Program Structure and Design, Prentice-Hall, Englewood Cliffs, N. J., 1975.

Capitolo 14

DEBUGGING E COLLAUDO

Come si è rilevato all'inizio del precedente capitolo, il debugging e il collaudo sono tra gli stadi dello sviluppo software, quelli che più necessitano di tempo per la loro esecuzione. **Anche se metodi quali la programmazione modulare, la programmazione strutturata, e la progettazione «top-down» possono semplificare i programmi e ridurre la frequenza degli errori, il debugging ed il collaudo sono tuttora difficoltosi poiché sono scarsamente definiti.** La selezione di una serie adeguata di dati di collaudo è di rado un procedimento scientifico o evidente. La scoperta degli errori sovente assomiglia al gioco «attacca la coda all'asino», con la differenza che l'asino si muove e il programmatore deve posizionare la coda con un telecomando. Senza dubbio, pochi lavori sono così frustranti come i programmi di «debugging».

Questo capitolo descriverà per prima cosa i mezzi a disposizione come supporto per il debugging. Quindi tratterà le procedure fondamentali del «debugging», descriverà i tipi comuni di errori, e presenterà alcuni esempi di debugging di programma. L'ultima parte descriverà come selezionare i dati di collaudo ed i programmi di test.

Non si farà nient'altro che descrivere gli scopi della maggior parte degli strumenti per il debugging. C'è assai poca standardizzazione in questo campo, e non sufficiente spazio per trattare tutti i dispositivi ed i programmi che sono attualmente disponibili. Gli esempi vi daranno delle idee sugli usi, i vantaggi, e le limitazioni dei particolari supporti hardware e software.

STRUMENTI SEMPLICI PER IL DEBUGGING

Questi sono gli strumenti a disposizione più semplici per il debugging:

- **Un metodo passo-passo.**
- **Un metodo con punto d'interruzione.**
- **Un programma di Scaricamento dei Registri (o utilità).**
- **Un programma di Scaricamento della Memoria.**

PASSO SINGOLO

Il metodo passo-passo permette di eseguire il programma un passo alla volta. La maggior parte dei microcomputer basati sullo Z80 hanno questa caratteristica, dato che la circuiteria è abbastanza semplice. Naturalmente, **le sole cose che si sarà in grado di osservare quando il computer esegue un passo-passo sono gli strati delle linee di uscita che si stanno controllando.** Le linee più importanti sono:

- Il Bus Dati.
- Il Bus Indirizzi.
- Le linee di controllo $\overline{MREQ}$ (Richiesta in Memoria), $\overline{IORQ}$ (Richiesta in Input/Output), $\overline{RD}$ (Lettura in Memoria), e $\overline{WR}$ (Scrittura in Memoria).

Se si controllano queste linee (o in hardware o in software), si potrà osservare l'avanzamento degli indirizzi, le istruzioni ed i dati come li esegue il programma. Si potrà dire che tipo di operazione sta compilando la CPU. Queste informazioni daranno l'opportunità di riconoscere degli errori quali istruzioni inesatte di Salto, indirizzi tralasciati o sbagliati, codici operativi errati, o valori di dati inesatti. Tuttavia, non si possono vedere i contenuti dei registri e dei flag senza l'aggiunta di ulteriori caratteristiche di «debugging» oppure una particolare sequenza d'istruzioni. Molte operazioni del programma non possono essere esaminate in tempo reale.

Tabella 14-1. Restart dello Z80 ed Indirizzi degli Interrupt

Istruzione o Ingresso Esterno (Cod. mnemonico) (Pin)	Codice Oggetto dell'Istruzione (Esadecimale)	Indirizzo di Destinazione (Esadecimale)
RST 00H	C7	0000
RST 08H	CF	0008
RST 10H	D7	0010
RST 18H	DF	0018
RST 20H	E7	0020
RST 28H	EF	0028
RST 30H	F7	0030
RST 38H or $\overline{INT}$ in Mode 1	FF	0038
$\overline{NMI}$		0066

LIMITI DEL MODO PASSO SINGOLO

Ci sono diversi errori che un modo passo-passo non può aiutare a scoprire. Questi comprendono errori di temporizzazione ed errori nei sistemi d'interrupt o DMA. Inoltre, il modo passo-passo è molto lento, dato che esegue tipicamente un programma a meno di un milionesimo della velocità dello stesso processore. Usando questo metodo con intervalli di un secondo, ci vorrebbero più di dieci giorni per eseguire il processo in tempo reale. Il modo passo-passo è utile soltanto per controllare la logica di brevi sequenze d'istruzioni.

PUNTO DI ARRESTO

Un punto d'arresto (breakpoint) è una zona nella quale il programma si ferma automaticamente o si pone in attesa cosicché l'utilizzatore può esaminare lo stato corrente del sistema. Il programma non riparte solitamente finché l'operatore non richiede una ripresa dell'esecuzione. I punti d'arresto permettono di controllare o di passare attraverso un intero pezzo di un programma. Pertanto, per vedere se una «routine» di inizializzazione è corretta, si può inserire un punto d'arresto al termine di questa e far scorrere il programma. Si possono così controllare le locazioni di memoria ed i registri per vedere se l'intero pezzo è corretto. Tuttavia, notate che se il pezzo di programma non è esatto, si dovrà ancora andare alla ricerca dell'errore, o con punti d'arresto più a monte o mediante un modo passo-passo.

I punti d'arresto sono di complemento al modo passo-passo. Si possono usare punti d'arresto o per localizzare l'errore o per passare attraverso porzioni di programma che si conoscono essere corrette. Si può quindi eseguire il debugging particolareggiato nel modo passo-passo. In qualche caso, i punti di arresto non influenzano la temporizzazione del programma; essi possono pertanto essere impiegati per controllare gli interrupt da input/output.

RST COME PUNTO D'ARRESTO

I punti d'arresto spesso utilizzano una parte o tutto il sistema d'interrupt del microprocessore. Alcuni processori hanno uno speciale INTERRUPT SOFTWARE oppure un ingresso TRAP che può agire come punto d'arresto. **Sullo Z80, se non si utilizzano già tutti i vettori RST nel programma, si può impiegare l'istruzione RST (RESTART) come punto d'arresto.** La Tabella 14-1 fornisce gli indirizzi di destinazione per le varie istruzioni RST. Il Capitolo 12 descrive la istruzione RST più in dettaglio. La routine del punto d'arresto può stampare il contenuto dei registri e della memoria oppure solo porsi in attesa (ad esempio eseguire HALT o un salto condizionato che dipende da un interruttore d'ingresso) finché non si concede al computer di proseguire. Se non si sta adoperando l'interrupt mascherabile ($\overline{INT}$) o l'interrupt non mascherabile ($\overline{NMI}$) nel sistema, si possono utilizzare quei vettori come punti d'arresto controllati dall'esterno. Ma ricordate che gli interrupt (compreso NMI) e RST fanno uso dello Stack e del Puntatore di Stack per conservare l'indirizzo di ritorno. La Figura 14-1 mostra una routine dove RST porta come conseguenza ad un ciclo senza fine. Si dovrebbe azzerare questo punto d'arresto con un RESET o con un segnale d'interrupt.

Figura 14-1. Una Semplice Routine con Punto di Arresto

```
        ORG     18H
RST18   EQU     18H
        JR      RST18   ;WAIT IN PLACE
```

Il metodo più semplice per inserire dei punti d'arresto è quello di sostituire il primo byte dell'istruzione con un'istruzione RST o sostituire l'istruzione stessa con una istruzione di Salto oppure di CALL. L'uso di un'istruzione RST è preferito sullo Z80, da che ciò comporta la sostituzione di un solo byte, mentre un JP o una CALL comportano tre byte. L'istruzione JR non è adatta al punto d'arresto poiché non si può garantire che il software di «debug» sia compreso entro -126 a +129 byte dell'istruzione che deve essere interrotta in un punto. Le istruzioni multi-byte impiegate per implementare i punti d'arresto possono comportare dei problemi sullo Z80 a causa della presenza di istruzioni ad un solo byte. Per illustrare questo programma, esaminate la porzione di programma di seguito riportata:

Indirizzo di Memoria (Esadecimale)	Contenuto di Memoria (Esadecimale)	Istruzione (Codice Mnemonico)		
100	7B		LD	A,E
101	87	L1:	ADD	A,A
102	87	L2:	ADD	A,A

Se si desidera sistemare un punto d'arresto alla locazione 100_{16} utilizzando una CALL o un JP a 3 byte, il codice presente nelle locazioni 101_{16} e 102_{16} verrà anch'esso ricoperto dall'istruzione CALL o JP. Ciò significa che colui che esegue il debugging deve fare attenzione al fatto che queste locazioni sono state modificate. Qualsiasi trasferimento di controllo ad L1 oppure ad L2 mentre viene sistemato il punto d'arresto produrrà dei risultati imprevisti a meno che non si preveda che l'operatore consideri questo caso. Questa sopraggiunta complicazione può essere evitata con l'uso di un'istruzione RST.

INSERIMENTO DI PUNTI DI ARRESTO

Molti programmi monitor hanno delle facilitazioni per l'inserimento e lo spostamento dei punti d'arresto implementati tramite qualche tipo d'istruzione di Salto. Tali punti d'arresto non agiscono sulla temporizzazione del programma fino a che non viene effettuato il punto d'arresto. Tuttavia, notate che questa procedura ncn funzionerà se una parte oppure tutto il programma è contenuto in ROM o PROM. Altri monitor implementano dei punti d'arresto controllando realmente le linee di indirizzo o il Contatore del Programma in hardware o in software. Questo metodo permette dei punti d'arresto sugli indirizzi in ROM o PROM, ma può influenzare la temporizzazione se l'indirizzo deve essere controllato in software. Una facilitazione più efficace permetterebbe all'utilizzatore di fare entrare un indirizzo al quale il processore trasferirebbe il controllo. Un'altra possibilità sarebbe un ritorno dipendente da un interruttore:

```
        ORG     18H
RST18   EQU     18H
        PUSH    AF              ;SALVA L'ACCUMULATORE, I FLAG
WAITS:  IN      A,(PIODRA)      ;PRENDI IL DATO DELL'INTERRUTTORE
        BIT     SW,A            ;L'INTERRUTTORE È CHIUSO?
        JR      NZ,WAITS        ;NO, ASPETTA FINCHÈ NON LO SIA
        POP     AF              ;RIMEMORIZZA L'ACCUMULATORE, I FLAG
        RET
```

Ricordate di riabilitare gli interrupt se la routine utilizza un ingresso d'interrupt esterno.

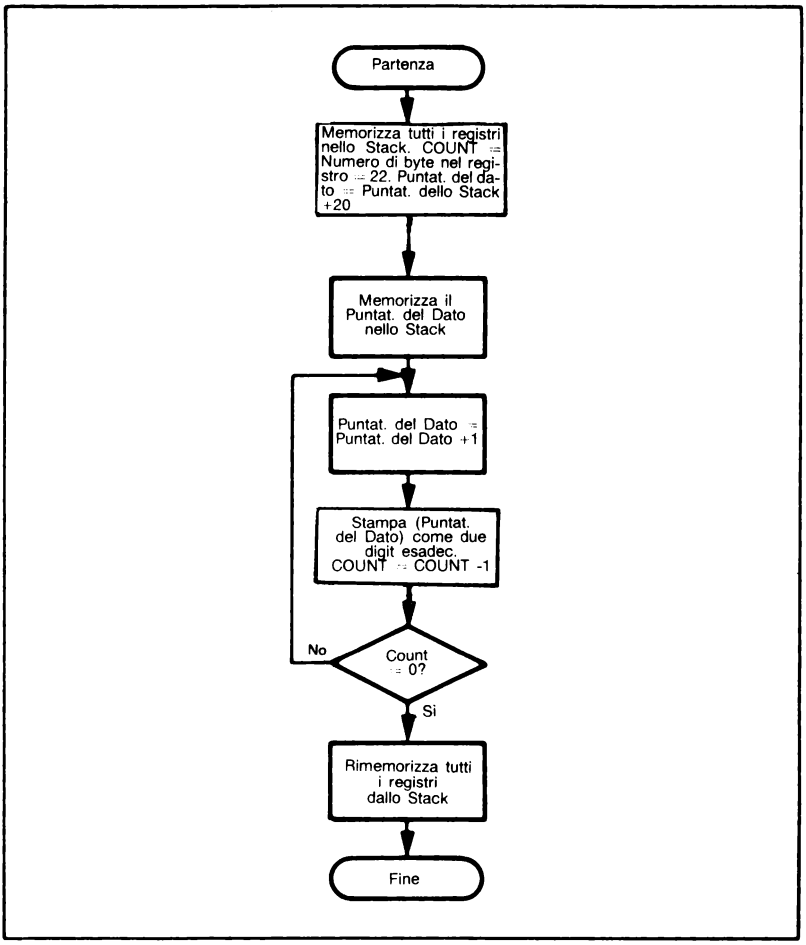


Figura 14-2. Diagramma di Flusso del Programma di Scarico di Registro

SCARICAMENTO DI REGISTRI

Un'utilità di Scaricamento dei Registri in un microcomputer è un programma che elenca il contenuto di tutti i registri della CPU. Quest'informazione non è solitamente ottenibile. **La seguente routine stamperà il contenuto di tutti i registri sulla stampante del sistema,** se si suppone che PRTHEX stampi il contenuto dell'Accumulatore sotto forma di due cifre esadecimali. La Figura 14-2 è un diagramma di flusso del programma e la Figura 14-3 rappresenta un risultato tipico. Supponiamo che la routine sia richiamata con un'istruzione CALL che memorizza il vecchio contenuto del Contatore del Programma in cima allo Stack.

```
;
;METTI IL CONTENUTO DI TUTTI I REGISTRI DELLA CPU NELLO
;   STACK (PURE PC È GIÀ NELLO STACK)
;
        PUSH    AF      ;SALVA I REGISTRI DELL'UTILIZZATORE REGOLARE
        PUSH    BC
        PUSH    DE
        PUSH    HL
        PUSH    IX      ;SALVA I REGISTRI INDICE
        PUSH    IY
        EX      AF,AF'  ;ACCEDI E SALVA I REGISTRI DELLA CPU CON APICE
        EXX
        PUSH    AF
        PUSH    BC
        PUSH    DE
        PUSH    HL
;
;USA IL PUNTATORE DELLO STACK COME INDIRIZZO DI PARTENZA
;
        LD      HL,0    ;PRENDI IL PUNTATORE DELLO STACK
        ADD     HL,SP
        LD      DE,20   ;CALCOLA IL PUNTATORE DELLO STACK ORIGINALE
        ADD     HL,DE
        PUSH    HL      ;SALVA IL PUNTATORE DELLO STACK ORIGINALE
                        ;   NELLO STACK
;
;STAMPA IL CONTENUTO DEI REGISTRI
;L'ORDINE È PC (ALTO), PC (BASSO), A,F,B,C,D,E,H,L,IX(ALTO),
;   IX (BASSO), IY (ALTO), IY (BASSO), A',F',B',C',D',E',H',L',SP(ALTO),SP(BASSO)
;
        LD      B,22    ;NUMERO DI BYTE = 22
PRNT1:  DEC     HL
        LD      A,(HL)  ;PRENDI UN BYTE DALLO STACK
        CALL    PRTHEX  ;E STAMPALO
        DJNZ    PRNT1
;
;RIMEMORIZZA I REGISTRI DALLO STACK
;
        POP     HL      ;ESEGUI UN POP E SCARTA IL PUNTATORE
                        ;   DI STACK ORIGINALE
        POP     HL      ;RIMEMORIZZA I REGISTRI CON APICE DELLA CPU
        POP     DE
        POP     BC
        POP     AF
        EX      AF,AF'
        EXX
        POP     IY      ;RIMEMORIZZA I REGISTRI INDICE
        POP     IX
        POP     HL      ;RIMEMORIZZA I REGISTRI REGOLARI DELLA CPU
        POP     DE
        POP     BC
        POP     AF
        RET
```

1D	(A)
42	(F)
07	(B)
3E	(C)
23	(D)
01	(E)
17	(H)
01	(L)
D3	(IX)
58	
E2	(IY)
A2	
36	(A')
67	(F')
E8	(B')
11	(C')
EB	(D')
09	(E')
D7	(H')
66	(L')
68	(STACK POINTER)
E2	

Figura 14-3. Risultati di un Tipico Scaricamento dei Registri dello Z80

SCARICAMENTO DI MEMORIA

Uno Scaricamento della Memoria è un programma che elenca il contenuto della memoria su un dispositivo di uscita (come ad esempio una stampante). Questo è un metodo assai più efficiente che osservare solamente le singole locazioni per esaminare delle matrici di dati o dei programma completi. Comunque, degli scaricamenti di memoria molto estesi non sono vantaggiosi (se si eccettua la fornitura di carta straccia) a causa della grande massa di informazioni che producono. Essi possono inoltre richiedere un lungo tempo di esecuzione su una stampante lenta. **Piccoli scaricamenti possono, tuttavia, fornire al programmatore una ragionevole quantità d'informazione che può essere esaminata come un'unità. Relazioni del tipo ripetizioni regolari di campioni di dati o «offset» di intere matrici possono diventare ovvie.**

Un generico scaricamento è sovente piuttosto difficile da scrivere. Il programmatore dovrebbe prestare attenzione alle seguenti situazioni:

1) La capacità dell'area di memoria supera i 256 byte, cosicché non sarà sufficiente un contatore ad 8 bit.
2) La locazione finale ha un indirizzo più piccolo della locazione di partenza. Questo può essere considerato come un errore, o può non provocare semplicemente alcun effetto in uscita, dato che l'utilizzatore potrebbe raramente desiderare di stampare tutti i contenuti della memoria in un insolito ordine.

Poiché la rapidità dello Scaricamento dipende dalla velocità del dispositivo d'uscita, raramente importa l'efficienza della routine. **Il seguente programma ignorerà i casi in cui l'indirizzo di partenza è maggiore di quello di arrivo, e tratterà dei blocchi di qualsiasi lunghezza.** Supponiamo che l'indirizzo di partenza sia contenuto nella Coppia di Registri DE e l'indirizzo di arrivo sia nella Coppia di Registri HL.

```
;
;FERMATI SE STAI FINENDO L'INDIRIZZO PRIMA DI INIZIARE L'INDIRIZZO
;
        AND     A           ;AZZERA IL CARRY
        SBC     HL,DE       ;STA FINENDO L'INDIRIZZO PRIMA DI PARTIRE?
        JR      C,DONE      ;SÌ, NON FARE IL DUMP DI NIENTE
        XCHG                ;METTI L'INDIRIZZO DI PARTENZA IN HL
        IN      DE          ;COUNT = NUMERO DI LOCAZIONI DELLE QUALI
                            ;        SI DEVE FARE DUMPED
;
;STAMPA IL CONTENUTO DELLE LOCAZIONI
;
DUMP:   LD      A,(HL)      ;PRENDI IL CONTENUTO DI UNA LOCAZIONE
        CALL    PRNT1       ;E STAMPALO
        INC     HL
        DEC     DE
        LD      A,E         ;È STATO FATTO IL DUMP DI TUTTE LE LOCAZIONI?
        OR      D
        JR      NZ,DUMP     ;NO, CONTINUA A FARE IL DUMP
DONE:   HALT
```

Notate che la sola istruzione di Sottrazione a 16 bit è SBC, che sottrae il contenuto di una coppia di registri e del Carry dalla Coppia di Registri HL. SBC, come le altre istruzioni di Sottrazione, posiziona ad 1 il Carry se è necessario un prestito (contrariamente a quanto dicono alcuni manuali dello Z80).

La Figura 14-4 rappresenta l'uscita ottenuta da uno scaricamento delle locazioni di memoria da 1000 a 101F.

23	1F	60	54	37	28	3E	00
6E	42	38	17	59	44	98	37
47	36	23	81	E1	FF	FF	5A
34	ED	BC	AF	FE	FF	27	02

Figura 14-4. Risultati di un Tipico Scarico di Memoria

Questa routine tratta in modo corretto il caso in cui le locazioni di partenza e d'arrivo sono le stesse (provate!). Dovrete interpretare attentamente i risultati se l'area che viene scaricata comprende lo Stack, dato che la «routine» di scaricamento stessa impiega lo Stack. PRNT 1 può inoltre modificare le locazioni di memoria e di Stack.

In uno scaricamento di memoria, i dati possono essere visualizzati in una quantità di modi diversi. Le forme comuni sono caratteri ASCII o coppie di cifre esadecimali per i valori ad 8 bit e quattro cifre esadecimali per i valori a 16 bit. Il formato dovrebbe essere scelto in base all'uso che si intende fare dello scaricamento. È quasi sempre più facile interpretare uno scaricamento di codice oggetto se esso viene visualizzato sotto forma esadecimale invece che ASCII.

Un formato comune e utile di scaricamento è di seguito rappresentato:

1000 54 68 65 20 64 75 6D 70 Lo scaricamento

Ciascuna linea consiste di tre parti. La linea inizia con l'indirizzo esadecimale del primo byte visualizzato sulla linea. Di seguito all'indirizzo ci sono otto o sedici byte visualizzati in forma esadecimale. L'ultima è la rappresentazione ASCII degli stessi otto o sedici byte. Provate a riscrivere il programma di scaricamento della memoria in modo che esso stampi l'indirizzo ed i caratteri ASCII come pure la forma esadecimale dei contenuti della memoria.

STRUMENTI PIÙ AVANZATI PER IL DEBUGGING

Gli strumenti più avanzati per il «debugging» che vengono più ampiamente usati sono:

- **Programmi simili per controllare il software.**
- **Analizzatori logici per controllare i segnali e la temporizzazione.**

Esistono molte varianti per ambedue questi strumenti, e si tratteranno solamente le caratteristiche tipiche.

SIMULATORE IN SOFTWARE

Il simulatore è l'equivalente computerizzato del computer «carta e matita». Esso **è un programma di un computer che passa attraverso il ciclo di funzionamento di un altro computer, non perdendo di vista il contenuto di tutti i registri, i flag, e le locazioni di memoria.** Si potrebbe, naturalmente, eseguire tutto questo a mano, ma ciò richiederebbe una grande quantità di lavoro e di cura dedicata agli esatti effetti di ciascuna istruzione. Il programma simulatore mai si stanca o resta confuso, dimentica una istruzione o un registro, o rimane senza carta.

La maggior parte dei simulatori sono vasti programmi FORTRAN. Essi possono essere acquistati o utilizzati sui servizi a divisione di tempo. Il simulatore dello Z80 è disponibile in parecchie versioni da sorgenti diverse.

Le caratteristiche tipiche di un simulatore sono:

1) Una opportunità per il punto d'arresto. Solitamente, i punti d'arresto possono essere sistemati dopo che un particolare numero di cicli è stato eseguito; quando si fa riferimento ad una locazione di memoria oppure ad una di una serie di locazioni di memoria, quando il contenuto di una locazione o di una appartenente ad una serie di locazioni viene modificato, o in altre condizioni.
2) Delle opportunità per lo scaricamento dei registri e della memoria che permettono di visualizzare i valori delle locazioni di memoria, dei registri, e delle porte I/0.
3) Una opportunità di tracciamento che stamperà il contenuto di registri o locazioni di memoria particolari ogni volta che il programma va a modificarli o usarli.
4) Una opportunità di caricamento che permette di posizionare dei valori all'inizio o variarli durante la simulazione.

Alcuni simulatori possono inoltre simulare l'input/output, gli interrupt e persino il DMA.

Il simulatore ha parecchi vantaggi:

1) Può fornire una descrizione completa dello stato del computer, dato che il programma simulatore non è vincolato dalle limitazioni dei «pin» o da altre caratteristiche della circuiteria sotto controllo.
2) Può fornire punti d'arresto, scaricamenti, tracciamenti, ed altre opportunità, senza utilizzare per niente lo spazio di memoria o il sistema di controllo del processore. Queste agevolazioni non interferiranno perciò con il programma dell'utilizzatore.
3) I Programmi, i punti di partenza, e altre condizioni sono facili da modificare.
4) Tutte le facilitazioni di un grande computer, comprese le periferiche e il software, sono disponibili per il progettista del microprocessore.

D'altra parte, il simulatore è limitato dalla sua base di software e dalla sua separazione dal microcomputer reale. Le principali limitazioni sono:

1) Il simulatore non può fornire aiuto nei problemi relativi alla temporizzazione, dato che esso funziona di gran lunga più lentamente del tempo reale e non emula l'hardware ed il software reali.
2) Il simulatore non può emulare completamente il gruppo dell'input/output.
3) Il simulatore è solitamente alquanto lento. La riproduzione di un secondo di tempo del processore reale può richiedere ore di tempo del computer. L'impiego del simulatore può risultare piuttosto dispendioso.

Il simulatore rappresenta la versione software del debugging; esso ha i tipici pregi e limiti di un metodo interamente basato sul software. Il simulatore può aiutare e penetrare nella logica del programma e negli altri problemi software, ma non può fornire alcun aiuto per quanto riguarda la temporizzazione, l'I/O, ed altri problemi hardware.

ANALIZZATORE LOGICO

L'analizzatore logico o a microprocessore è la soluzione dal punto di vista hardware per il debugging. Fondamentalmente, l'analizzatore è la versione digitale parallelo dell'oscilloscopio più classico. L'analizzatore visualizza l'informazione in binario, esadecimale o in forma mnemonica su un CRT, ed ha una varietà di eventi d'innesco, di soglie, e d'ingressi. La maggior parte degli analizzatori posseggono pure una memoria in modo tale che essi possono visualizzare i contenuti passati dei bus.

La procedura tipica è quella di predisporre un evento di innesco, come l'avvento di un particolare indirizzo sul Bus Indirizzi e d'una istruzione sul Bus Dati. Per esempio, si potrebbe innescare l'analizzatore se il microcomputer cerca di memorizzare dei dati in un particolare indiriz-

zo o di eseguire un'istruzione di ingresso o di uscita. Si può pertanto osservare la sequenza di eventi che hanno preceduto il punto d'arresto. **Si possono scoprire in questo modo i soliti inconvenienti compresi i picchi di rumore (o «glitches»), sequenze di segnali inesatte, sovrapposizione di forme d'onda, e altri errori nei segnali e nella temporizzazione. Naturalmente, non si potrebbe impiegare un simulatore del software per diagnosticare quegli errori; niente di più conveniente di un analizzatore logico potrebbe essere utilizzato per scoprire gli errori nella logica del programma.**

CARATTERISTICHE IMPORTANTI DEGLI ANALIZZATORI LOGICI

Gli analizzatori logici variano sotto molti aspetti. Alcuni di questi sono:

1) Numero delle linee d'ingresso. Ne sono necessarie almeno 24 per controllare un Bus Dati ad 8 bit ed un Bus Indirizzi a 16 bit. Ne sono necessarie ancora di più per i segnali di controllo, i clock ed altri ingressi importanti.
2) Capacità di memoria. Ogni stato precedente che viene salvato occuperà parecchi byte.
3) Frequenza massima. Deve essere di diversi MHz per trattare con i processori più veloci.
4) Minima ampiezza del segnale (importante per cogliere i «glitches»).
5) Tipo e numero degli eventi d'innesco ammessi. Caratteristiche importanti sono i ritardi di pre-innesco e di post-innesco; questi permettono all'utilizzatore di visualizzare degli eventi che accadono prima o dopo l'evento d'innesco.
6) Metodi di collegamento al microcomputer. Questo potrebbe richiedere un'interfaccia piuttosto complessa.
7) Numero di canali di visualizzazione.
8) Visualizzatori binari, esadecimali o mnemonici.
9) Formati dei visualizzatori.
10) Registri del tempo di mantenimento dei segnali.
11) Capacità della sonda.
12) Soglie singole o doppie.

Tutti questi fattori sono importanti quando si confrontano diversi analizzatori logici e per microprocessore, dato che questi strumenti sono nuovi e non standardizzati. Un'enorme varietà di prodotti è già disponibile e questa varietà diventerà ancora più grande nel futuro.

Gli analizzatori, naturalmente, **sono necessari solamente per sistemi con temporizzazioni complesse. Le applicazioni semplici con periferiche a bassa velocità hanno pochi problemi hardware che il progettista non riesca a gestire mediante un oscilloscopio classico.**

DEBUGGING CON LISTE DI CONTROLLO

Può darsi che il progettista non riesca a controllare un intero programma manualmente, tuttavia, vi sono dei punti critici che il progettista può agevolmente controllare. **Si può usare il controllo sistematico effettuato normalmente per mettere in evidenza un grande numero di errori senza ricorrere a nessun strumento di «debugging».**

CHE COSA INCLUDERE NELLA LISTA DI CONTROLLO

Il problema consiste nel dove orientare lo sforzo. La risposta è: nei punti che possono essere trattati con una risposta si-no o con un semplice calcolo aritmetico. Non cercate di eseguire operazioni aritmetiche complesse, seguite tutti i flag, o mettete alla prova tutti i casi concepibili. Limitate il vostro controllo manuale ad argomenti che possono essere facilmente affrontati. Lasciate che i problemi complessi siano risolti mediante il supporto degli strumenti per debugging. Non procedete in modo sistematico; costruitevi la lista di controllo, ed accertatevi che il programma esegua correttamente le operazioni fondamentali.

La prima fase consiste nel confrontare il digramma di flusso o altra documentazione di programma con il vero codice. Accertatevi che ogni cosa che compare in uno compare anche

nell'altro. Una semplice lista di controllo eseguirà il compito. È facile dimenticare completamente un salto o una parte di processo.

Successivamente concentratevi sui cicli del programma. Accertatevi che tutti i registri e le locazioni di memoria utilizzati nei cicli vengano visualizzati correttamente. Questa è una comune origine di errori; ancora una volta, basterà una semplice lista di controllo.

Ora osservate ciascun salto condizionato. Selezionate un caso campione che produrrebbe un salto ed uno che non lo produrrebbe; provateli entrambi. Il salto è corretto o invertito? Se il salto implica un controllo se un numero è superiore o inferiore a una soglia, provate il caso dell'uguaglianza. Avviene il salto corretto? Accertatevi che la vostra scelta sia coerente con la definizione del problema.

Osservate i cicli nell'insieme. Provate manualmente la prima e l'ultima iterazione; queste costituiscono spesso dei particolari casi problematici. Cosa avviene se il numero di ripetizioni è zero; cioè non ci sono dati o la tabella non ha elementi? Il programma salta nel modo dovuto? I programmi spesso eseguiranno inutilmente una iterazione o, anche peggio, decrementeranno i contatori oltre lo zero prima di controllarli.

Verificate ogni cosa fino ad arrivare all'ultima affermazione. Non supponete (in modo ottimistico) che il primo errore sia il solo errore presente nel programma. Il controllo manuale vi permetterà di ottenere il massimo beneficio dalle prove di debugging, dato che vi libererete in anticipo di molti errori semplici.

DOMANDE PER CONTROLLO MANUALE

Di seguito è riportato un breve riassunto dei quesiti relativi al controllo manuale:

1) Ogni elemento della progettazione del programma è presente nel programma (e viceversa per scopi di documentazione)?
2) Tutti i registri e le locazioni di memoria utilizzati nei cicli vengono inizializzati prima del loro uso?
3) Sono corretti tutti i salti condizionati?
4) Tutti i cicli iniziano e terminano correttamente?
5) I casi d'uguaglianza vengono trattati in modo corretto?
6) I casi insignificanti vengono trattati correttamente?

RICERCA DEGLI ERRORI

ERRORI COMUNI

Naturalmente, malgrado tutte queste precauzioni (o se ne trascurate qualcuna) i programmi spesso non funzionano ancora. Il progettista viene lasciato col problema di come trovare gli errori. La lista di controllo manuale fornisce un punto di partenza se non è stato precedentemente impiegato; alcuni errori che possono non essere stati eliminati sono:

1) **Trascurare d'inizializzare le variabili quali contatori, puntatori, somme ecc.** Non supponete che i registri, le locazioni di memoria, o i flag contengano necessariamente zero prima del loro utilizzo.

2) **Invertire la logica di un salto condizionato,** come l'utilizzo di un Salto su Carry quando si vorrebbe invece eseguire un Salto su Non Carry. Ricordate gli effetti di un confronto o di una sottrazione (A è il contenuto dell'Accumulatore, M il contenuto del registro o della locazione di memoria):

$$\begin{aligned} \text{flag Zero} &= 1 \quad \text{se } A = M \\ &= 0 \quad \text{se } A \neq M \\ \text{flag Carry} &= 1 \quad \text{se } A < M \\ &= 0 \quad \text{se } A \geq M \end{aligned}$$

Notate in particolare che il Carry è uguale a 0 se A = M, (caso di uguaglianza). Pertanto, Salto su Carry significa salto se A < M, e Salto su Non Carry significa salto se A ≥ M. Se d'altra parte desiderate il caso di uguaglianza, provate a invertire il ruolo di A ed M oppure ad aggiungere 1 ad M. Per esempio, se si vuole che avvenga un salto nel caso A ≥ 10, usare:

```
CP      10
JR      NC,ADDR
```

Se, d'altronde, si vuole che avvenga un salto nel caso A > 10, usare:

```
CP      11
JR      NC,ADDR
```

3) **Aggiornare i contatori e puntatori nel punto sbagliato o non eseguire affatto questa operazione.** Assicuratevi che non vi sia nessun percorso attraverso un ciclo che salti o ripeta le istruzioni di aggiornamento.

4) **Non fallire giustamente nei casi insignificanti** quali mancanza di dati in un «buffer», mancanza di prove da eseguire, o assenza d'ingressi in transazione. Non supporrete che tali casi non capitino mai a meno che il programma non li escluda specificatamente.

Ci si deve attendere la comparsa di altri problemi quali:

5) **Invertire l'ordine degli operandi.** Ricordate che l'istruzione LD sposta il secondo operando nel primo operando. Per esempio LD B,A sposta il contenuto di A in B, non il contrario.

6) **Modificare i flag di condizione prima del loro utilizzo.** Ricordate che INC e DEE, se applicati ad un singolo registro o locazione di memoria, influenza tutti i flag eccetto il Carry. Ricordate inoltre che POP AF ed EX AF,AF' agiscono su tutti i flag, e che le istruzioni Logiche azzerano il Carry.

7) **Trascurare di modificare i flag di condizione quando si intende invece variarli.**
I flag Zero e Segno possono non rappresentare lo stato corrente dell'Accumulatore, dato che molte istruzioni (in particolare LD) non modificano i flag. Notate che l'esecuzione di incrementi o di decrementi su coppie di registri (per esempio, INC HL o DEC BC) e di complementi dell'Accumulatore (CPL) non agisce affatto su alcun flag.

8) **Far confusione tra valori e indirizzi.**
Ricordate che LD, HL, 1000H carica HL con il numero 1000 (esadecimale) mentre LD HL, (1000H) carica HL con il contenuto delle locazioni di memoria 1000 e 1001. Una simile distinzione si applica a LD A,COUNT e LD A,(COUNT).

9) **Ri-inizializzare accidentalmente un registro o una locazione di memoria.**
Assicuratevi che nessuna istruzione di Salto trasferisca a ritroso il controllo a delle dichiarazioni relative all'inizializzazione.

10) **Far confusione tra numeri e caratteri.**
Ricordate che le rappresentazioni ASCII ed EBCDIC di cifre sono diverse dalle cifre stesse. Per esempio, ASCII 7 è 37 esadecimale, mentre 07 esadecimale è il carattere ASCII BELL.

11) **Far confusione tra numeri binari e decimali.**
Ricordate che la rappresentazione BCD di un numero differisce dalla sua rappresentazione binaria. Per esempio, BCD 36, se viene trattato come una semplice costante decimale, è equivalente a 54 decimale (verificatelo).

12) **Invertire l'ordine nella sottrazione.** Prestate attenzione alle altre operazioni (come la divisione) che non commutano. Ricordate che SUB e CP eseguono A-M, non M-A.

13) **Trascurare gli effetti di subroutine e di macro.**
Non supponete che le chiamate di subroutine o di macro non modifichino i flag, i registri, o le locazioni di memoria. Assicurativi degli esatti effetti delle subroutine e dei macro. Notate che è assai importante documentare questi effetti in modo tale che l'utilizzatore possa determinarli senza dover scorrere l'intera lista.

14) **Utilizzare erroneamente delle istruzioni di Spostamento (Shift).**
Ricordate l'esatto effetto di RLC, RL, RRC, RR, SLA, SRA, e SRL. Essi sono tutti spostamenti ad 1 bit. SLA e SRL azzerano ambedue il bit evacuato. SRA conserva il segno (il bit più significativo) estendendolo alla sua destra, RLC e RRC sono spostamenti circolari che non coinvolgono il Carry nel registro circolare; RL e RR sono spostamenti circolari che coinvolgono il Carry. Ricordate che queste istruzioni influenzano tutti i flag, anche nel caso vengano applicate ai dati contenuti in una locazione di memoria. Notate, tuttavia, che gli spostamenti ad una parola RLCA, RLA, RRCA e RRA agiscono soltanto sul Carry.

15) **Contare in modo inesatto la lunghezza di una matrice.**
Ricordate che ci sono cinque (e non quattro) locazioni di memoria incluse negli indirizzi da 0100 a 0104, compreso.

16) **Far confusione tra registri e coppie di registri.**
Ricordate che i registri e le coppie di registri della CPU sono fisicamente gli stessi. Essi si possono utilizzare singolarmente per dati ad 8 bit o in coppie per indirizzi o dati a 16 bit, ma non ambedue nello stesso tempo. Notate che INC HL incrementa in realtà L, agendo su H solo se L viene incrementato fino a raggiungere il valore zero.

17) **Far confusione tra registri ad 8 bit ed a 16 bit.**
L'Accumulatore ed i registri della CPU hanno una lunghezza di otto bit, mentre i registri indice, il Contatore del Programma, il Puntatore di Stack, e le copie di registri sono lunghi 16 bit. Non si può trasferire il contenuto di un registro a 16 bit in un registro ad 8 bit o viceversa.

18) **Dimenticarsi del fatto che i numero o gli indirizzi a 16 bit occupano due locazioni di memoria.**
LD HL,(40H) carica la Coppia di Registri HL con il contenuto delle locazioni di memoria 0040 e 0041. Allo stesso modo, PUSH DE memorizza la Coppia di Registri DE in due locazioni di Stack. Ricordate inoltre che lo Z80 immagazzina tutte le quantità a 2 byte nel formato basso ordine/alto ordine. Per esempio, LD(40H),HL memorizzerà il contenuto del Registro L nella locazione 0040 e il contenuto del Registro H nella locazione 0041.

19) **Far confusione tra Stack e Puntatore di Stack.**
DEC, INC ed LD influenzano il Puntatore di Stack, non il contenuto dello Stack.
PUSH e POP trasfericono dati verso e dallo Stack. Ricordate che anche CALL, RET, RETI RETN, e RST utilizzano lo Stack per salvare o ripristinare il Contatore del programma. La risposta a un interrupt implica sempre il salvataggio del vecchio contenuto del Contatore del Programma nello Stack anche se non viene ottenuta esternamente nessuna istruzione esplicita (come nella risposta a $\overline{NMI}$ o a $\overline{INT}$ nei modi d'interrupt 1 o 2). Notate che le istruzioni del tipo EX (SP), HL non influenzano il Puntatore di Stack; esse scambiano le due locazioni di memoria in cima allo Stack con il contenuto di una coppia di registri o con il contenuto di un registro Indice, ma lasciano invariata la lunghezza dello Stack.

20) **Dimenticare d'inizializzare il Puntatore di Stack.**
Ricordate che si deve collocare l'indirizzo di memoria esatto nel Puntatore di Stack prima di eseguire la chiamata di una qualsiasi subroutine o di compiere una qualsiasi operazione sullo Stack.

21) **Modificare il contenuto di un registro o di una locazione di memoria prima del loro utilizzo.**
Ricordate che LD modifica il contenuto della destinazione (ma non della sorgente). Fate attenzione alle istruzioni che utilizzano implicitamente certi registri – per esempio DJNZ decrementa il registro B; LDI, LDIR, LDD, LDDR, LPI, CPIR, CPD, e CPDR decrementano tutte il Contatore di Byte presente nella Coppia di registri BC ed incrementano o decrementano la Coppia di Registri DE. Le istruzioni INI, INIR, IND, INDR, OUTI, OUTIR, OUTD, e OTDR decrementano tutte il registro B ed incrementano o decrementano la Coppia di Registri HL.

22) **Dimenticare di trasferire il controllo in punti posti al di là di sezioni di programma che non dovrebbero essere eseguite in situazioni particolari.**
Ricordate che il computer procederà in maniera sequenziale attraverso la memoria del programma a meno che non venga specificatamente ordinato di fare altrimenti.

DEBUGGING DI PROGRAMMI GESTITI AD INTERRUPT

Il debugging di programmi pilotati ad interrupt è particolarmente difficile, dato che possono verificarsi casualmente degli errori. Se, per esempio, il programma abilita agli interrupt un po' d'istruzioni in anticipo, si verificherà un errore solo se viene ricevuto un interrupt mentre il programma sta eseguendo quelle poche istruzioni. **Infatti si può solitamente supporre che il verificarsi casuale degli errori sia provocato dal sistema d'interrupt. Gli errori tipici nei programmi pilotati ad interrupt sono i seguenti:**

1) **Dimenticare di riabilitare gli interrupt dopo averne accettato e servito uno.** Il processore disabilita il sistema d'interrupt automaticamente dopo l'operazione di RESET o dopo aver accettato un interrupt. Assicuratevi che nessuna sequenza possibile trascuri la riabilitazione del sistema d'interrupt. Ricordate che, oltre che la riabilitazione degli interrupt, il programma deve sovente compiere delle azioni per portare all'azzeramento del segnale d'interrupt. Se ciò non viene eseguito, tutto apparirà come se il dispositivo interrompente stia costantemente sollecitando il servizio.

2) **Utilizzare l'Accumulatore prima del suo salvataggio,** cioè PUSH AF deve precedere qualsiasi operazione d'ingresso o di uscita che coinvolga l'Accumulatore.

3) **Dimenticare di salvare e ripristinare l'Accumulatore ed i flag** (Coppia di Registri AF).

4) **Ripristinare i registri nell'ordine sbagliato:**
Se l'ordine nel quale essi erano stati salvati era:

```
PUSH AF
PUSH BC
PUSH DE
PUSH HL
```

l'ordine per il loro ripristino dovrebbe essere:

```
POP HL
POP DE
POP BC
POP AF
```

5) **Abilitare gli interrupt prima di stabilire tutte le condizioni necessarie** come la priorità, i flag, le configurazioni dei PIO e dei SIO, i puntatori, i contatori, ecc. In questo caso può tornare utile un listing di controllo.

6) **Lasciare i risultati nei registri e distruggerli nella fase di ripristino.**
Come è stato precedentemente osservato, i registri non dovrebbero essere usati per passare le informazioni tra il programma vero e proprio e le routine di servizio degli interrupt.

7) **Dimenticarsi del fatto che RST (e $\overline{NMI}$) lascia un indirizzo nello Stack sia che lo si usi oppure no.**

8) **Non disabilitare gli interrupt durante i trasferimenti multibyte o sequenze di istruzioni.**
Attendetevi in particolare delle situazioni in cui la routine di servizio dell'interrupt può utilizzare le stesse locazioni di memoria che sta utilizzando il programma.

Questi elenchi dovrebbero aver fornito almeno qualche idea in merito a dove ricercare gli errori. Purtroppo, anche il metodo di debugging più sistematico può ancora lasciare dei problemi veramente imbarazzanti, in particolare quando vengono coinvolti gli interrupt[3].

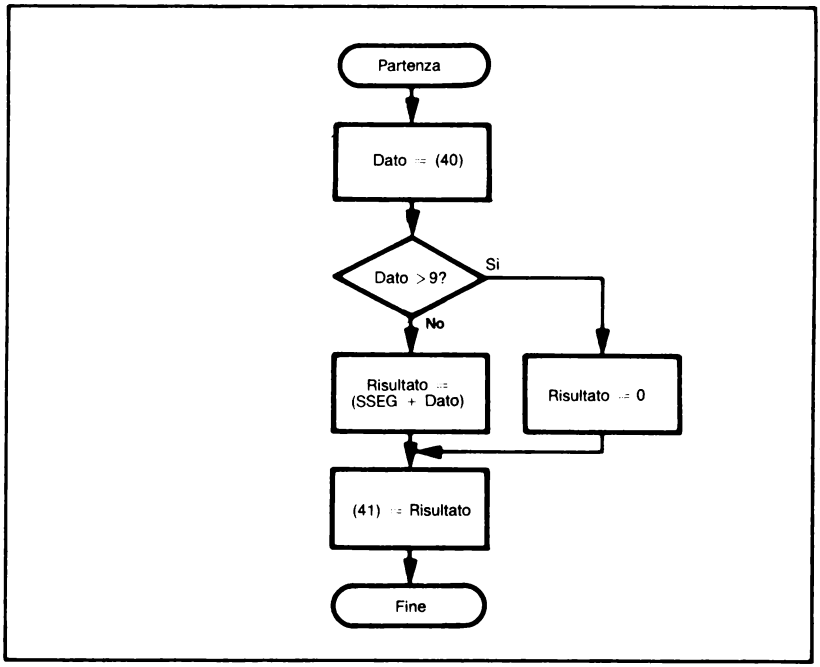


Figura 14-5. Diagramma di Flusso della Conversione da Decimale a 7 Segmenti

Esempio 1 di Debugging: Conversione da Decimale a Sette-Segmenti

DEBUGGING DI UN PROGRAMMA DI CONVERSIONE DI CODICE

Il programma converte un numero decimale contenuto nella locazione di memoria 0040 in un codice a sette-segmenti da porre nella locazione 0041. Esso porta allo spegnimento del visualizzatore se la locazione di memoria 0040 non contiene un numero decimale.

Programma Iniziale (dal diagramma di flusso in Figura 14-5):

```
        LD      A,40H       ;PRENDI IL DATO
        CP      9           ;IL DATO È UN DIGIT DECIMALE?
        JR      C,DONE      ;NO, MANTIENI IL CODICE DI ERRORE
        LD      HL,(ISSEG)  ;PRENDI L'INDIRIZZO DI BASE
                            ;   DELLA TABELLA A 7 SEGMENTI
        LD      D,A
        ADD     HL,DE       ;TROVA L'ELEMENTO MEDIANTE INDICIZZAZIONE
        LD      A,(HL)      ;PRENDI IL CODICE A 7 SEGMENTI DALLA TABELLA
DONE:   LD      (41H),A     ;SALVA IL CODICE A 7 SEGMENTI
                            ;   O IL CODICE DI ERRORE
        HALT
SSEG:   DEFB    3FH
        DEFB    06H
        DEFB    5BH
        DEFB    4FH
        DEFB    66H
        DEFB    6DH
        DEFB    7DH
        DEFB    07H
        DEFB    7DH
        DEFB    6FH
```

Utilizzando la procedura con lista di controllo, si è in grado di trovare i seguenti errori:

1) Il blocco che ha azzerato il Risultato era stato omesso.
2) Il salto condizionato era inesatto.

Per esempio, se il dato è pari a zero, CP 9 posiziona ad 1 il Carry, dato che $0 < 9$. Tuttavia, il salto sulla condizione opposta (cioè JR NC, DONE) non produrrà ancora il risultato corretto. Ora il programma tratta il caso di uguaglianza in modo errato dato che, se il dato è 9, CP 9 azzera il Carry e provoca un salto. La versione corretta è:

```
        CP      10          ;IL DATO È UNA CIFRA DECIMALE?
        JR      NC,DONE     ;NO, ACCETTA IL CODICE DI ERRORE
```

Secondo Programma:

```
        LD      B,0         ;PRENDI IL CODICE DI ERRORE
                            ;   PER SPEGNERE IL DISPLAY
        LD      A,40H       ;PRENDI IL DATO
        CP      10          ;IL DATO È UN DIGIT DECIMALE?
        JR      NC,DONE     ;NO, MANTIENI IL CODICE DI ERRORE
        LD      HL,(SSEG)   ;PRENDI L'INDIRIZZO DI BASE
                            ;   DELLA TABELLA A 7 SEGMENTI
        LD      D,A
        ADD     HL,DE       ;TROVA L'ELEMENTO MEDIANTE INDICIZZAZIONE
        LD      A,(HL)      ;PRENDI IL CODICE A 7 SEGMENTI DALLA TABELLA
DONE:   LD      (41H),A     ;SALVA IL CODICE A 7 SEGMENTI
                            ;   O IL CODICE DI ERRORE
        HALT
SSEG:   DEFB    3FH
        DEFB    06H
        DEFB    5BH
        DEFB    4FH
        DEFB    66H
        DEFB    6DH
        DEFB    7DH
        DEFB    07H
        DEFB    7DH
        DEFB    6FH
```

Questa versione è stata controllata con successo manualmente. Poiché il programma era semplice, lo stadio successivo era di procedere passo-passo lungo lo stesso programma mediante dei dati letti. I dati selezionati come prova erano:

0	(il numero più piccolo)
9	(il numero più grande)
10	(un caso limite)
6B (esad.)	(casuale)

La prima prova era con lo zero nella locazione 0040 (esadecimale). Il primo errore era ovvio – LD A,40H ha caricato il numero 40 in A, non il contenuto della locazione di memoria 0040. L'istruzione corretta era LD A,(40H). Eseguita questa correzione, il programma ha proceduto senza alcun errore apparente finché ha cercato di eseguire l'istruzione LD A,(HL).

Il contenuto del Bus Indirizzi durante il prelievo del dato era 0647, un indirizzo che neppure esiste nel microcomputer. Chiaramente, qualcosa non aveva funzionato.

Era ora tempo di eseguire qualche ulteriore controllo manuale. Dato che si sapeva che JR NC, DONE era corretto, l'errore era presenta al di là di quell'istruzione ma prima di LD A,(HL). Un controllo manuale ha rivelato quanto segue:

1) LD HL,(SSEG) pone 3F (esadecimale) in L e 06 (esadecimale) in H.
   Ciò è chiaramente sbagliato. Si vuole LD HL<SSEG e non LD HL, (SSEG). Cioè, si vuole che l'indirizzo SSEG, e non il contenuto presente a quell'indirizzo, venga caricato nella Coppia di Registri HL.
2) LD D,A pone il valore 0 nel registro D.
   Ciò è errato – il dato va collocato in E, dato che si vuole sommarlo ai bit meno significativi dell'indirizzo della tabella. Infatti, un'istruzione dovrebbe azzerare il Registro D, dato che il programma errato non stava inizializzando affatto o variando l'altra metà della Coppia di Registri DE.

Terzo Programma:

```
        LD      B,0         ;PRENDI IL CODICE DI ERRORE
                            ;   PER SPEGNERE IL DISPLAY
        LD      A,(40H)     ;PRENDI IL DATO
        CP      10          ;IL DATO È UN DIGIT DECIMALE?
        JR      NC,DONE     ;NO, CONSERVA IL CODICE DI ERRORE
        LD      HL,SSEG     ;PRENDI L'INDIRIZZO DI BASE
                            ;   DELLA TABELLA A 7 SEGMENTI
        LD      E,A
        LD      D,0         ;UTILIZZA IL DATO COME INDICE A 16 BIT
        ADD     HL,DE       ;TROVA L'ELEMENTO MEDIANTE INDICIZZAZIONE
        LD      A,(HL)      ;PRENDI IL CODICE A 7 SEGMENTI DALLA TABELLA
DONE:   LD      (41H),A     ;SALVA IL CODICE A 7 SEGMENTI
                            ;   O IL CODICE DI ERRORE
        HALT
SSEG:   DEFB    3FH
        DEFB    06H
        DEFB    5BH
        DEFB    4FH
        DEFB    66H
        DEFB    6DH
        DEFB    7DH
        DEFB    07H
        DEFB    7DH
        DEFB    6FH
```

Questo programma ha prodotto i seguenti risultati:

Dato	Risultato
00	3F
09	6F
0A	0A
6B	6B

Il programma non stava azzerando il risultato se il dato non era valido, cioè maggiore di 9. Il programma non ha mai utilizzato il codice di spegnimento contenuto nel Registro B. Dato che il programma era semplice, si sarebbe potuto provare per tutte le cifre decimali. I risultati sono stati i seguenti:

Dato	Risultato
0	3F
1	06
2	5B
3	4F
4	69
5	6D
6	7D
7	07
8	7D
9	6F

Notate che il risultato per il numero 8 è inesatto – dovrebbe essere 7F. Dato che tutto il resto è corretto, l'errore è quasi sicuramente nella tabella. Infatti l'ingresso 8 nella tabella è stato copiato male.

Il programma finale è:

```
;
;CONVERSIONE DA DECIMALE A 7 SEGMENTI
;
        LD      B,0         ;PRENDI IL CODICE DI ERRORE
                            ;   PER SPEGNERE IL DISPLAY
        LD      A,(40H)     ;PRENDI IL DATO
        CP      10          ;IL DATO È UN DIGIT DECIMALE?
        JR      NC,DONE     ;NO, CONSERVA IL CODICE DI ERRORE
        LD      HL,SSEG     ;PRENDI L'INDIRIZZO DI BASE
                            ;   DELLA TABELLA A 7 SEGMENTI
        LD      E,A
        LD      D,0         ;UTILIZZA IL DATO COME INDICE A 16 BIT
        ADD     HL,DE       ;TROVA L'ELEMENTO MEDIANTE INDICIZZAZIONE
        LD      B,(HL)      ;PRENDI IL CODICE A 7 SEGMENTI DALLA TABELLA
DONE:   LD      A,B
        LD      (41H),A     ;SALVA IL CODICE A 7 SEGMENTI
                            ;   O IL CODICE DI ERRORE
        HALT
SSEG:   DEFB    3FH
        DEFB    06H
        DEFB    5BH
        DEFB    4FH
        DEFB    66H
        DEFB    6DH
        DEFB    7DH
        DEFB    07H
        DEFB    7FH
        DEFB    6FH
```

Gli errori incontrati in questo programma sono tipicamente quelli che i programmatori in linguaggio «assembly» dello Z80 dovrebbero prevedere. Essi comprendono:

1) Trascurare di inizializzare registri e locazioni di memoria.
2) Invertire la logica nei salti condizionati.
3) Eseguire salti in modo non corretto nel caso di operandi uguali.
4) Far confusione tra indirizzamenti diretti e immediati, cioè fra dati e indirizzi.
5) Sbagliare nel distinguere tra dati ad 8 bit e indirizzi a 16 bit.
6) Eseguire un salto in un punto sbagliato in modo che un percorso attraverso il programma risulti inesatto.
7) Copiare elenchi di numeri (oppure istruzioni) in modo scorretto.

Notate che le istruzioni immediate come ADD, SUB, AND, ecc. raramente presentano dei problemi. Un errore particolarmente fastidioso che vi dovreste aspettare è l'inversione degli operandi nelle istruzioni LD. Molti di questi errori possono essere eliminati con l'uso di un linguaggio di programmazione di sistema a basso livello come il PLS/ASM[4].

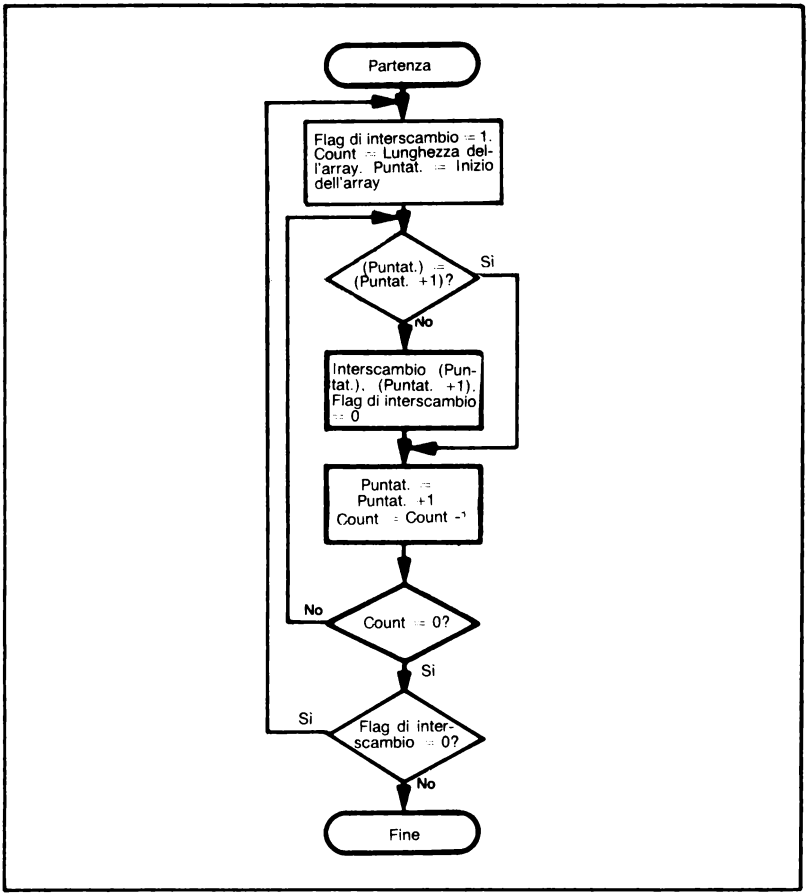


Figura 14-6. Diagramma di Flusso di un Programma di Ricerca

Esempio di Debugging: Classificazione in Ordine Decrescente

DEBUGGING DI UN PROGRAMMA DI CLASSIFICAZIONE

Il programma classifica una matrice di numeri binari ad 8 bit privi di segno in ordine descrescente. La matrice inizia nella cella di memoria 0041 e la sua lunghezza è contenuta nella cella 0040.

Programma Iniziale (dal diagramma di flusso in Figura 14-6):

```
        LD      C,0         ;AZZERA IL FLAG DI SCAMBIO
        LD      A,(40H)     ;COUNT = LUNGHEZZA DELL'ARRAY
        LD      C,A
        LD      HL,41H      ;PUNTA ALL'INIZIO DELL'ARRAY
PASS1:  LD      A,(HL)      ;PRENDI UN ELEMENTO DALL'ARRAY
        INC     HL
        CP      (HL)        ;È MINORE DELL'ELEMENTO SUCCESSIVO?
        JR      C,CNT       ;NO, NON È NECESSARIO NESSUN SCAMBIO
        LD      (HL),A      ;SÌ, SCAMBIA GLI ELEMENTI
        INC     HL
CNT:    DJNZ    PASS1
        DEC     C           ;È STATO POSIZIONATO IL FLAG DI SCAMBIO?
        JR      NZ,PASS1    ;SÌ, FAI UN ALTRO PASSO
        HALT
```

Il controllo manuale dimostra che tutti i blocchi nel diagramma di flusso sono stati implementati nel programma e che tutti i registri sono stati inizializzati. I salti condizionati debbono essere esaminati attentamente. L'istruzione JR C,CNT deve forzare un salto se il nuovo valore è minore o uguale al valore vecchio. Notate che il caso di ugaglianza non deve dare come risultato uno scambio, dato che questo creerà un ciclo senza fine con i due elementi uguali che si spostano avanti ed indietro.

Provate con un esempio:

```
(0040) = 30
(0041) = 37
```

CP (HL) si risolve nel calcolo di 30-37. Il Carry viene posizionato ad uno. Questo esempio dovrebbe dare come risultato uno scambio ma ciò in realtà non si verifica.

JR NC,CNT fornirà il salto appropriato in questo caso. Se i due numeri sono uguali, il confronto azzererà il Carry e JR NC,CNT è ancora corretto.

E che dire di JR NZ,SORT posta al termine del programma? Se ci sono degli elementi non in ordine, il flag di scambio sarà uno, cosicché il salto risulta errato. Dovrebbe essere JR Z,SORT.

Ora controlliamo manualmente la prima iterazione del programma. L'inizializzazione porta ai seguenti risultati:

```
A  =  COUNT
B  =  COUNT
C  =  0
HL =  0041
```

Gli effetti delle istruzioni cicliche sono:

```
        LD      A,(HL)    ;A=(0041)
        INC     HL        ;HL=0042
        CP      (HL)      ;(0041)-(0042)
        JR      NC,CNT
        LD      (HL),A    ;(0042)=(0041)
        INC     HL        ;HL=0043
CNT:    DJNZ    PASS1     ;B=COUNT-1
```

Notate che si sono già controllate le istruzioni di Salto Condizionato. Chiaramente la logica è scorretta. Se i primi due numeri non sono in ordine, il risultato dopo la prima iterazione dovrebbe essere:

```
(0041) = OLD (0042)
(0042) = OLD (0041)
    HL = 0042
     B = COUNT-1
```

Invece, essi sono:

```
(0041) = IMMUTATO
(0042) = OLD (0041)
    HL = 0043
     B = COUNT-1
```

L'errore in HL è facile da correggere. Il secondo INC HL è inutile e si dovrebbe omettere. Lo scambio richiede un po' più di attenzione ed un registro temporaneo, cioè:

```
LD      D,(HL)
LD      (HL),A
DEC     HL
LD      (HL),D
INC     HL
```

Uno scambio richiede sempre un luogo per la memorizzazione temporanea nel quale un numero può essere salvato mentre l'altro è in fase di trasferimento.

Tutte queste variazioni necessitano di una ritrascrizione del programma, cioè,

```
        LD      C,0         ;AZZERA IL FLAG DI SCAMBIO
        LD      A,(40H)     ;COUNT = LUNGHEZZA DELL'ARRAY
        LD      C,A
        LD      HL,41H      ;PUNTA ALL'INIZIO DELL'ARRAY
PASS1:  LD      A,(HL)      ;PRENDI UN ELEMENTO DALL'ARRAY
        INC     HL
        CP      (HL)        ;È MINORE DELL'ELEMENTO SUCCESSIVO?
        JR      NC,CNT      ;NO, NON È NECESSARIO NESSUN SCAMBIO
        LD      D,(HL)      ;SÌ, SCAMBIA GLI ELEMENTI
        LD      (HL),A
        DEC     HL
        LD      (HL),D
        INC     HL
CNT:    DJNZ    PASS1
        DEC     C           ;È STATO POSIZIONATO IL FLAG DI SCAMBIO?
        JR      NZ,PASS1    ;SÌ, FAI UN ALTRO PASSO
        HALT
```

E che dire dell'ultima iterazione? Diciamo che vi sono tre elementi:

```
(0040) = 03
(0041) = 02
(0042) = 04
(0043) = 06
```

Ogni volta che ripassa, il programma incrementa di uno la Coppia di Registri HL. Così, all'inizio della terza iterazione,

$$(HL) = 0041 + 2 = 0043$$

Gli effetti delle istruzioni cicliche sono:

```
LD      A,(HL)      ;A = (0043)
INC     HL          ;HL = 0044
CP      (HL)        ;(0043) - (0044)
```

Ciò è scorretto; il programma ha provato a spostarsi oltre la parte finale dei dati. La precedente iterazione avrebbe dovuto essere, infatti l'ultima, dato che il numero di coppie è inferiore di uno rispetto al numero di elementi. La correzione consiste nel ridurre il numero di iterazioni di un'unità; ciò si può realizzare ponendo DEC B dopo LD A,(40H).

Che dire dei casi insignificanti? Cosa avviene se la matrice non contiene affatto nessun elemento, o solamente un elemento? La risposta è che il programma non funziona in maniera corretta e può modificare erroneamente un intero blocco di dati e senza alcun preavviso (sperimentate tutto ciò!). Le correzioni per il trattamento dei casi insignificanti sono semplici ma essenziali; il prezzo da pagare è costituito solamente da pochi byte di memoria onde evitare dei problemi che potrebbero poi rivelarsi assai difficoltosi da risolvere.

Il nuovo programma e:

```
        LD      C,0         ;AZZERA IL FLAG DI SCAMBIO
        LD      A,(40H)     ;COUNT = LUNGHEZZA DELL'ARRAY
        CP      2           ;L'ARRAY HA 2 O PIÙ ELEMENTI?
        JR      C,DONE      ;NO, NON È NECESSARIA NESSUNA AZIONE
        LD      B,A
        DEC     B           ;NUMERO DI COPPIE = COUNT-1
        LD      HL,41H      ;PUNTA ALL'INIZIO DELL'ARRAY
PASS1:  LD      A,(HL)      ;PRENDI UN ELEMENTO DALL'ARRAY
        INC     HL
        CP      (HL)        ;È MINORE DELL'ELEMENTO SUCCESSIVO?
        JR      NC,CNT      ;NO, NON È NECESSARIO NESSUN SCAMBIO
        LD      D,(HL)      ;SÌ, SCAMBIA GLI ELEMENTI
        LD      (HL),A
        DEC     HL
        LD      (HL),D
        INC     HL
CNT:    DJNZ    PASS1
        DEC     C           ;È STATO POSIZIONATO IL FLAG DI SCAMBIO?
        JR      NZ,PASS1    ;SÌ, FAI UN ALTRO PASSO
        HALT
```

Ora è il caso di controllare il programma sul computer o sul simulatore. Una semplice serie di dati è la seguente:

```
(0040) = 02
(0041) = 00
(0042) = 01
```

Questa serie consiste di due elementi posti nell'ordine sbagliato. Il programma necessiterebbe di due passaggi. Il primo passaggio dovrebbe riordinare gli elementi, producendo:

```
(0041) = 01
(0042) = 00
     C = 01
```

Il secondo passaggio dovrebbe completare l'operazione e produrre:

```
     C = 00
```

Questo programma è piuttosto lungo per il metodo passo-passo, e perciò si utilizzerà invece il metodo con punti d'arresto. Ciascun punto d'arresto bloccherà il computer e stamperà il contenuto di tutti i registri. I punti d'arresto verranno:

1) Dopo LD HL, 41H per controllare l'inizializzazione.
2) Dopo CP (HL) per controllare il confronto.
3) Dopo il secondo INC HL (cioè proprio prima della label CNT) per controllare lo scambio.
4) Dopo DEC C per controllare il compimento di un passaggio attraverso la matrice. Il contenuto dei registri dopo il primo punto d'arresto era:

Registri	Contenuto
A	02
B	01
C	00
H	00
L	41

Questi sono tutti corretti, e dunque il programma sta eseguendo in questo caso l'inizializzazione in modo preciso.

I risultati in corrispondenza del secondo punto d'arresto erano:

Registri	Contenuto
A	00
B	01
C	00
H	00
L	42
CARRY	1

Questi risultati sono pure corretti. I risultati in corrispondenza del terzo punto d'arresto erano:

Registri	Contenuto
A	00
B	01
C	00
D	01
H	00
L	42

Controllando la memoria rappresentata:

(0041) = 01
(0042) = 00

I risultati in corrispondenza del quarto punto d'arresto erano:

Registri	Contenuto
A	00
B	01
C	00
D	01
H	00
L	42

Qui, il Registro C non contiene il corretto valore – esso avrebbe dovuto essere posizionato ad uno per indicare che è avvenuto uno scambio. Infatti, un'occhiata rivolta al programma rivela che nessuna istruzione modifica mai C per indicare lo scambio. La correzione consiste nel porre l'istruzione LD C,1 dopo JR NC,CNT.

Ora la procedura è quella di caricare il Registro C con il valore corretto e di continuare. La seconda iterazione del secondo punto d'arresto fornisce:

Registri	Contenuto
A	00
B	00
C	00
H	00
L	43
CARRY	1

Chiaramente il programma ha proseguito in modo errato senza la reinizializzazione dei registri (in particolare HL). Il salto condizionato che dipende dal flag di scambio dovrebbe trasferire il controllo fino in fondo all'inizio del programma, e non alla label PASS1.

La versione finale del programma è:

```
SORT:   LD      C,0         ;AZZERA IL FLAG DI SCAMBIO
        LD      A,(40H)     ;COUNT = LUNGHEZZA DELL'ARRAY
        CP      2           ;L'ARRAY HA 2 O PIÙ ELEMENTI?
        JR      C,DONE      ;NO, NON È NECESSARIA NESSUNA AZIONE
        LD      B,A
        DEC     B           ;NUMERO DI COPPIE = COUNT-1
        LD      HL,41H      ;PUNTA ALL'INIZIO DELL'ARRAY
PASS1:  LD      A,(HL)      ;PRENDI UN ELEMENTO DALL'ARRAY
        INC     HL
        CP      (HL)        ;È MINORE DELL'ELEMENTO SUCCESSIVO?
        JR      NC,CNT      ;NO, NON È NECESSARIO NESSUN SCAMBIO
        LD      C,1         ;SÌ, POSIZIONA IL FLAG DI SCAMBIO
        LD      D,(HL)      ;SCAMBIA GLI ELEMENTI
        LD      (HL),A
        DEC     HL
        LD      (HL),D
        INC     HL
CNT:    DJNZ    PASS1
        DEC     C           ;È STATO POSIZIONATO IL FLAG DI SCAMBIO?
        JR      NZ,SORT     ;SÌ, FAI UN ALTRO PASSO
        HALT
```

Chiaramente non si possono controllare tutti i possibili valori d'ingresso per questo programma. Due altre semplici serie di dati a scopo di debugging sono:

1) Due elementi uguali

```
(0040) = 02
(0041) = 00
(0042) = 00
```

2) Due elementi già in ordine decrescente

```
(0040) = 02
(0041) = 01
(0042) = 00
```

INTRODUZIONE AL COLLAUDO

UTILIZZO DI PROCESSI DI COLLAUDO NEL DEBUGGING

Il collaudo di programma (testing) è in stretta relazione col debugging di programma. Certamente alcuni dei casi di collaudo saranno i medesimi dati di test utilizzati per il debugging, quali:

- **Casi insignificanti come assenza di dati o presenza di un unico elemento.**
- **Casi particolari che il programma sceglie per qualche motivo.**
- **Esempi semplici che usano particolari parti del programma.**

Nel caso del programma di conversione da decimale a sette-segmenti, questi casi comprendono tutte le possibili situazioni. I dati di collaudo consistono di:

- I numeri da 0 a 9.
- Il caso limite 10.
- Il caso casuale 6N.

Il programma non distingue nessun altro caso. **In questo caso il debugging e il collaudo sono virtualmente gli stessi.**

Nel programma di classificazione, il problema è più difficile. Il numero di elementi potrebbe essere compreso tra 0 e 255, e ciascun elemento potrebbe giacere in quel campo di variazione. Il numero di casi possibili è perciò enorme. Inoltre, il programma è moderatamente complesso. Come scegliere i dati di collaudo che daranno un grado di sicurezza in quel programma? **In questo caso il collaudo richiede delle decisioni in fase di progettazione.** Il problema del collaudo è particolarmente difficile se il programma dipende da sequenze di dati in tempo reale. Come scegliere i dati, generarli, e presentarli al microcomputer in modo realistico?

AUSILI PER IL COLLAUDO

La maggior parte degli strumenti precedentemente menzionati per eseguire il debugging sono utili anche per il collaudo. Gli analizzatori logico o per microprocessore possono aiutare a controllare l'hardware; i simulatori possono aiutare nel controllo del software. Altri strumenti possono anche essere di aiuto, come ad esempio,

1) **Simulazioni di I/O** che possono simulare una varietà di dispositivi partendo da un dispositivo con un solo ingresso ed una sola uscita.
2) **Emulatori nel circuito** che permettono di collegare il prototipo a un sistema di sviluppo o ad un pannello di controllo e di provarlo.
3) **Simulatori di ROM** che hanno la flessibilità di una RAM ma la temporizzazione della particolare ROM o PROM che verrà impiegata nel sistema finale.
4) **Sistemi funzionanti in tempo reale** che possono fornire ingressi oppure interrupt in istanti specifici (o forse casualmente) ed indicare la comparsa di uscita. Si possono anche includere tracce e punti d'arresto in tempo reale.
5) **Emulazioni** (sovente su computer microprogrammabili) che possono fornire rapidità di esecuzione in tempo reale ed I/O programmabile.[5]
6) **Interfacce** che consentono a un altro computer di controllare il sistema dell'I/O e di collaudare il programma del microcomputer.
7) **Programmi di collaudo** che verificano ogni diramazione in un programma per il controllo degli errori logici.
8) **Programmi di generazione per il collaudo** che possono produrre dei dati a caso o altre distribuzioni.

Esistono dei teoremi formali per il collaudo, ma essi sono solitamente applicabili soltanto a programmi molto brevi.

Si deve ricordare che l'apparecchiatura di collaudo non invalida la prova modificando le condizioni al contorno. Spesso, l'apparecchiatura di test può eseguire il buffering, memorizzare o condizionare segnali d'ingresso e d'uscita. Il sistema reale può non praticare tutto ciò, e comportarsi quindi in maniera del tutto diversa.

Inoltre, il software supplementare nelle condizioni di prova può utilizzare dello spazio di memoria o parte del sistema d'interrupt. Esso può anche fornire la preservazione dell'errore ed altre prestazioni che non esistono nel sistema finale. Un supporto per il collaudo software deve essere realizzato proprio come il corrispondente supporto di collaudo hardware, dato che gli inconvenienti software possono essere critici proprio come quelli hardware.

Le emulazioni e le simulazioni non sono, naturalmente, mai precise. Esse sono di solito adatte alla verifica della logica ma possono raramente fornire un aiuto nel collaudo dell'interfaccia o della temporizzazione. D'altra parte, le attrezzature di test in tempo reale non forniscono molto di più di una supervisione della logica di programma e possono influenzare l'interfacciamento e la temporizzazione.

SCELTA DEI DATI DI COLLAUDO

Pochissimi programmi reali possono essere verificati per tutti i casi. Il progettista deve scegliere una serie campione che in qualche modo riesca a descrivere l'intero campo di possibilità.

COLLAUDO STRUTTURATO

Il collaudo dovrebbe far, naturalmente, parte della totale procedura di sviluppo. La progettazione «top-down» e la programmazione strutturata prevedono il collaudo come parte della progettazione stessa. Questo viene detto collaudo strutturato.[6] Ciascun modulo entro un programma strutturato dovrebbe essere verificato separatamente. **Il collaudo, come pure la progettazione, dovrebbero essere modulari, strutturati, e del tipo «top-down».**

COLLAUDO DI PROCESSI SPECIALI

Ma ciò lascia in sospeso la questione sulla scelta dei dati di collaudo per un modulo. Il progettista deve per prima cosa elencare tutti i casi particolari che un programma riconosce. Questi possono comprendere:

- Casi insignificanti.
- Casi di uguaglianza.
- Situazioni particolari.

I dati di collaudo debbono comprendere tutti questi casi sopraelencati.

FORMAZIONE DI CLASSI DI DATI

Si deve successivamente identificare ciascuna classe di dati che le dichiarazioni all'interno del programma possono distinguere. Queste possono comprendere:

- Numeri positivo o negativi.
- Numeri al di sopra o al di sotto di una soglia particolare.
- Dati che includono oppure no una particolare sequenza o caratteri.
- Dati che sono presenti oppure no in un particolare istante.

Se i moduli sono brevi, il numero complessivo di classi dovrebbe essere ancora piccolo anche se ciascuna divisione è moltiplicativa, cioè due divisioni a due vie portano come risultato a quattro classi di dati.

Si debbono ora separare le classi a secondo se il programma produce un risultato diverso per ciascun ingresso nella classe (come in una tabella) o se produce lo stesso risultato per ciascun ingresso (come un avvertimento del fatto che un parametro è al di sopra di una soglia). A seconda della scelta si può includere ciascun elemento se il numero complessivo è piccolo o campionare se il numero è grande. Il campionamento dovrebbe comprendere tutti i casi limite ed almeno un caso scelto in maniera casuale. Le tabelle dei numeri casuali sono disponibili in libri, ed i generatori di numeri casuali fanno parte delle facilitazioni che posseggono la maggior parte dei computer.

SCELTE DI DATI DALLE CLASSI

Si deve badare alle distinzioni che possono non essere chiare. Per esempio, lo Z80 considererà un numero ad 8 bit privo di segno che è maggiore di 127 come un numero negativo; tutto questo si deve considerare quando si impiegano le istruzioni di Salto che dipendono dal bit di Segno. Si devono inoltre tenere presenti le istruzioni che non influenzano i flag, che producono overflow nell'aritmetica con i segni, e le distinzioni tra quantità che indicano indirizzi (16 bit) e quantità che indicano dati (8 bit).

Esempio 1 di Collaudo: Programma di Classificazione

COLLAUDO DI UN PROGRAMMA DI CLASSIFICAZIONE

I casi particolari qui sono chiari:

- Assenza di elementi nella matrice.
- Presenza di un elemento, la cui grandezza può essere scelta in modo casuale.

L'altro caso particolare che deve essere considerato è quello nel quale gli elementi sono uguali.

Ci potrebbe essere qualche problema in questo caso con i segni e con la lunghezza dei dati. Notate che la stessa matrice deve contenere meno di 256 elementi. L'uso dell'istruzione LD C,1 oppure SET 1,C invece di DEC C per azzerare il flag di scambio significa che non ci sarà alcuna difficoltà se il numero di elementi o di scambi supera 128.

Si potrebbero controllare gli effetti del segno dividendo a metà i casi di prova regolari con numeri di elementi compresi tra 128 e 255 e compresi tra 2 e 127. Tutti i valori andrebbero scelti casualmente in modo da evitare il più possibile delle predisposizioni inconscie.

Esempio 2 di Collaudo: Numeri di Autocontrollo (vedi Capitolo 8)

COLLAUDO DI UN PROGRAMMA ARITMETICO

Si supporrà in questo caso che un precedente controllo di validità ha accertato che il numero ha la giusta lunghezza e consiste di cifre valide. Dato che il programma non fa altre distinzioni, i dati di collaudo dovrebbero essere scelti in maniera casuale. Qui si dimostrerà ideale una tabella di numeri casuali o un generatore di questi numeri; il campo di variazione dei numeri casuali è da 0 a 9.

PRECAUZIONI NEL COLLAUDO

REGOLE PER IL COLLAUDO

Il progettista può semplificare gli stadi di collaudo realizzando dei programmi in modo pratico. Si dovrebbero utilizzare le seguenti regole:

1) Cercare di eliminare i casi insignificanti prima possibile senza introdurre delle distinzioni superflue.
2) Minimizzare il numero di casi particolari. Ciascuno di questi vuole dire tempo di debugging e di collaudo supplementare.
3) Considerare le validità dell'esecuzione oppure i controlli di errore sui dati prima di iniziare il processo.
4) Stare attenti alle distinzioni inutili e involontarie, particolarmente quando si trattano numeri provvisti di segno o si utilizzano operazioni che si riferiscono a numeri provvisti di segno.
5) Verificare manualmente i casi limite. Questi costituiscono spesso una fonte di errori. Assicurarsi che la definizione del problema specifichi cosa avviene in questi casi.
6) Rendere il programma il più generale possibile in modo ragionevole. Ogni distinzione e routine separata aumenta il collaudo necessario.
7) Dividere il programma e progettare i moduli in modo che il collaudo possa proseguire a passi in congiunzione con gli altri stati di sviluppo del software.[7]

CONCLUSIONI

Il debugging ed il collaudo sono i figliastri del processo di sviluppo del software. La maggior parte dei progetti lascia troppo poco tempo a loro disposizione e la maggior parte dei libri di testo li trascura. Ma i progettisti e i manager trovano sovente che questi stadi sono i più costosi e necessitano di una grande quantità di tempo. Il progresso può essere molto difficile da misurare o produrre. Il software di debugging e di collaudo di un microprocessore è particolarmente difficile poiché gli strumenti potenti per l'hardware e il software che si possono impiegare nei computer più grandi sono raramente disponibili per i microcomputer.

Il progettista dovrebbe realizzare il debugging ed il collaudo con attenzione. Si raccomanda la seguente procedura:

1) Cercare di scrivere dei programmi che possono essere facilmente sottoposti al debugging ed al collaudo. La programmazione modulare, la programmazione strutturata, e la progettazione «top-down» si rivelano tecniche utili.
2) Preparare un piano di debugging e di collaudo come parte della progettazione del programma. Decidere per tempo quali dati si debbono generare e quale apparecchiatura sarà necessaria.
3) Eseguire il debugging ed il collaudo su ciascun modulo come parte del processo di progettazione «top-down».
4) Eseguire il debugging sulla logica di ciascun modulo in maniera sistematica. Utilizzare i listing di controllo, i punti d'arresto ed il modo passo-passo. Se la logica del programma risulta complessa, considerare l'utilizzazione del simulatore del software.
5) Controllare sistematicamente la temporizzazione di ogni modulo se ciò costituisce un problema. Un oscilloscopio può risolvere molti inconvenienti se si pianifica il collaudo in modo appropriato. Se la temporizzazione è complessa, considerare l'utilizzazione di un analizzatore logico o per microprocessore.
6) Assicurarsi che i dati di test siano dei campioni rappresentativi. Tenere presente le qualsiasi classi di dati che il programma può distinguere. Includere tutti i casi insignificanti e particolari.
7) Se il programma tratta ciascun elemento in maniera diversa o se il numero di casi è grande, scegliere i dati di collaudo in maniera casuale.[8]
8) Registrare tutti i risultati della prova come parte della documentazione. Se sorgono dei problemi, non si dovranno ripetere i casi di collaudo che sono già stati verificati.

BIBLIOGRAFIA

1. Per ulteriori informazioni suqli analizzatori logici, vedere:
   R.L. Down, "Understanding Logic Analyzers", Computer Design, Giugno 1977, pp. 188-191.
   W.A. Farnbach, "Bring up Your μP", Electronic Design, 10 Luglio 1976, pp. 80-85.
   B. Farly, "Logic Analyzers Aren't All Alike". Electronic Design, 1 Febbraio 1978, pp. 70-76.
   K. Pines, "What Do Logic Analyzers Do?", Digital Design, Settembre 1977, pp. 55-77.
   N.A. Robin, "The Logic Analyzer: A Computer Troubleshooting Tool", Computer Design, Marzo 1976, pp. 89-96.
   S. Runyon, "Focus on Logic and μP Analyzers", Electronic Design, 1 Febbraio 1977, pp. 40-50.
   A. Santoni, "The Latest Logic Analyzers Offer More Functions and Less Cost". Electronic Design, 1 Febbraio 1978, pp. 26-32.

2. Vedere W.J. Weller, Assembly Level Programming for Small Computers, Lexington Books, Lexington, Mass., 1975.

3. Alcune direttive per i problemi relativi al debugging degli interrupt sono fornite in R. L. Baldrige, "Interrupts Add Power, Complexity to Microcomputer System Design", EDN, 5 Agosto 1977, pp. 67-73.

4. Vedere C. Bass, "PLZ: A Family of System Programming Languages for Microprocessors", Computer, Marzo 1978, pp. 34-39.

5. Vedere, per esempio H.R. Burris, "Time-Scaled Emulations of the 8080 Microprocessor", Proceedings of the 1977 National Computer Conference, pp. 937-946.

6. Vedere D.A. Walsh, "Structured Testing", Datamation, Luglio 1977, pp. 111-118.

7. Il collaudo (ed il debugging) sono anche trattati in R.A. DeMillo et al., "Hints on Test Data Selection: Help for the Practicing Programmer", Computer, Aprile 1978, pp. 34-41 and in W.F. Dalton. "Design Microcomputer Software", Electronics, 19 Gennaio, 1978, pp. 97-101.

8. I numeri casuali e la loro generazione sono trattati in T.G. Lewis, Distribution Sampling for Computer Simulation, Lexington Books, Lexington, Mass., 1975 and in R.A. Mueller, et al., "A Random Number Generator for Microprocessors", Simulation, Aprile 1977, pp. 123-127.

Capitolo 15

DOCUMENTAZIONE E RIPROGETTAZIONE

Un programma funzionante non è il solo requisito di sviluppo del software. Pure una adeguata documentazione è una parte importante di un prodotto di software. La documentazione non solo aiuta il progettista nelle fasi di collaudo e di debugging, ma è anche essenziale per un uso successivo e per un ampliamento del programma. Un programma scarsamente documentato sarà difficile da mantenere o da espandere.

Può succedere che un programma usi troppa memoria o che sia eseguito troppo lentamente. Il progettista deve allora considerare dei modi per migliorarlo. Questa fase è chiamata riprogettazione e richiede che vi concentriate sulle parti del programma che possono dare la maggiore miglioria.

PROGRAMMI CHE SI AUTO-DOCUMENTANO

REGOLE PER PROGRAMMI CHE SI AUTODOCUMENTANO

Sebbene nessun programma non si autodocumenti mai completamente, possono essere utili **alcune regole** menzionate precedentemente. Esse **comprendono:**

- Una struttura chiara e semplice con il minor numero possibile di trasferimenti di controllo (salti).
- Uso di nomi e di label significative.
- Uso di nomi per dispositivi di I/O, per parametri, fattori numerici, ecc.
- Dare risalto alla semplicità piuttosto che ad un minor risparmio nell'uso di memoria, al tempo di esecuzione o alla stampa.

Per esempio, il programma seguente invia una stringa di caratteri ad una telescrivente:

```
        LD      A,(2000H)
        LD      B,A
        LD      HL,1000H
W:      LD      A,(HL)
        OUT     (6),A
        CALL    XXX
        INC     HL
        DJNZ    W
        HALT
```

Anche senza commenti possiamo migliorare il programma come segue:

```
MESSG   EQU     1000H
COUNT   EQU     2000H
TTYSIO  EQU     6
        LD      A,(COUNT)
        LD      B,A
        LD      HL,MESSG
OUTCH:  LD      A,(HL)
        OUT     (TTYSIO),A
        CALL    BITDLY
        INC     HL
        DJNZ    OUTCH
        HALT
```

Questo programma è sicuramente più facile da capire di quello precedente. Anche senza ulteriore documentazione, potreste probabilmente indovinare la funzione del programma e il significato di gran parte di variabili. **Altre tecniche di documentazione non possono sostituire la autodocumentazione.**

Qualche altra nota sulla scelta dei nomi:

SCELTA DI NOMI UTILI

1) **Usare un nome chiaro** quando è disponibile, come TTY e CRT per dispositivi di uscita, START o RESET per indirizzi, DELAY o SORT per sottoprogrammi, COUNT o LENGHT per dati.
2) **Evitare acronomi** come S16BA per SORT 16-BIT ARRAY. Questi solo raramente hanno significato per qualcuno.
3) **Usare parole complete** o quasi quando è possibile, come DONE, PRINT, SEND, ecc.
4) **Tenere** il più possibile **distinti i nomi.**

COMMENTI

La forma più evidente di documentazione aggiuntiva è il commento. Tuttavia, pochi programmi (anche quelli usati come esempi nei libri) hanno commenti effettivi. Dovreste considerare le seguenti indicazioni per ottenere buoni commenti.

INDICAZIONI PER I COMMENTI

1) **Non ripetete il significato del codice di istruzione.** Piuttosto, spiegate lo scopo dell'istruzione nel programma. Commenti come

```
DEC B                   ;B=B-1
```

non aggiungono nulla alla documentazione. Usate piuttosto

```
DEC B                   ;NUMERO DI LINEA=NUMERO DI LINEA-1
```

Ricordatevi che conoscete il significato dei codici operativi e che chiunque altro può vederli in un manuale. **Il punto importante è di spiegare quale compito sta svolgendo il programma.**

2) **Fate i commenti più chiari possibile.** Non usate abbreviazioni o acronomi senza che essi non siano ben noti (come ASCII, PIO o UART) o standard (come no per numero, ms per millisecondo, ecc). Evitate commenti come:

```
DEC B                   ;LN=LN-1
        oppure
DEC B                   ;DEC LN BY 1
```

La stampa in più non è tutto ciò che costa.

3) **Commentate ogni punto importante od oscuro.** Fate particolarmente attenzione a mettere in evidenza operazioni che possono non avere funzioni evidenti, quali:

```
AND 11011111B        ;SPEGNERE IL LETTORE DI NASTRO
```

oppure

```
ADD HL,DE            ;INDICE DELLA TABELLA PER IL CODICE GRAY
```

Le operazioni di I/O spesso richiedono commenti più ampi. Se non siete pienamente sicuri di ciò che fa un'istruzione o se dovete pensarci su aggiungete un commento di chiarimento. Questi, più tardi, vi farà risparmiare tempo e sarà molto utile per la documentazione.

4) **Non commentate le cose ovvie.** Un commento su ogni linea rende semplicemente difficile il trovare i punti importanti. Sequenze tipiche come:

```
INC     HL
DJNZ    RICERCA
```

non richiedono di essere rimarcate a meno che non stiate facendo qualcosa di speciale. Un solo commento sarà spesso sufficiente per parecchie linee, come:

```
RRCA            ;DIGIT DI SCAMBIO
RRCA
RRCA
RRCA
LD      A,C     ;SCAMBIO DEI BYTE PIÙ SIGNIFICATIVI CON I BYTE
LD      C,B     ;   MENO SIGNIFICATIVI
LD      B,A
```

5) **Porre i commenti sulle linee alle quali essi si riferiscono o all'inizio di una sequenza.**

6) **Mantenere aggiornati i vostri commenti.** Se cambiate un programma, cambiate i commenti.

7) **Usate termini e forme standard nei commenti.** Non preoccupatevi delle ripetizioni. Nomi diversi per le stesse cose creano confusione, anche se le differenze sono come COUNT e COUNTER, START e BEGIN, DISPLAY e LED o PANEL e SWITCHES.
Non c'è alcun guadagno nel non essere consistenti. Le variazioni vi possono sembrare ovvie ora, ma più tardi possono non essere chiare; altre potranno essere confuse con quelle iniziali.

8) **Fare commenti mescolati con brevi istruzioni.** Dare una completa spiegazione ai commenti principali e all'altra documentazione. Altrimenti il programma si perde nei commenti e può diventare arduo anche e trovare il programma stesso.

9) **Perfezionate i vostri commenti.** Se vi imbattete in un commento che non potete leggere o copiare, cambiatelo. Se vi accorgete che il listing sta aumentando, aggiungete qualche linea di spazi. I commenti non si perfezioneranno da soli; infatti essi peggioreranno allorché lascerete indietro il compito e dimenticherete ciò che avete fatto esattamente.

10) **Prima di ogni sezione, sottosezione, o subroutine di maggiore importanza, inserite numerosi commenti che descrivano le funzioni del codice che segue.** Si deve fare attenzione a descrivere tutti gli ingressi, tutte le uscite e gli effetti secondari, come pure l'algoritmo utilizzato.

11) È buona norma, **quando si modificano programmi funzionanti, usare commenti per indicare la data, l'autore ed il tipo della modifica effettuata.**

Ricordatevi che i commenti sono importanti. Buoni commenti vi faranno risparmiare tempo e fatica. Dedicate un po' di lavoro ai commenti e cercate di renderli i più effettivi possibile.

Esempio 1 di come fare commenti: Addizione in Precisione Multipla

Il programma basilare è:

```
        LD      A,(30H)
        LD      B,A
        LD      HL,41H
        LD      DE,51H
        AND     A
ADDWD:  LD      A,(DE)
        ADC     A,(HL)
        LD      (HL),A
        INC     DE
        INC     HL
        DJNZ    ADDWD
        HALT
```

Commentare dapprima i punti importanti. Questi consistono principalmente in inizializzazioni, fetch di dati ed operazioni di processo. Non preoccupatevi delle sequenze standard come puntatori di aggiornamento e contatori. Ricordate che i nomi sono più chiari dei numeri, cosicché usateli liberamente.

La nuova versione del programma è:

```
;ADDIZIONE IN MULTI-PRECISIONE
;
;QUESTO PROGRAMMA REALIZZA UNA ADDIZIONE MULTI-BYTE
;
;INGRESSI:    LOCAZIONE 30H = LUNGHEZZA DEI NUMERI (IN BYTE)
;             LOCAZIONI 41H-50H = PRIMO ADDENDO IN ORDINE LSB→MSB
;             LOCAZIONI 51H-60H = SECONDO ADDENDO
;USCITE:      LOCAZIONI 41H-51H = SOMMA
;
LENGHT  EQU     30H
NUMB1   EQU     41H
NUMB2   EQU     51H
        LDA     LENGHT      ;COUNT = LUNGHEZZA DEI NUMERI (IN BYTE)
        LD      B,A
        LD      HL,NUMB1    ;PARTI DAI BIT LSB DEL PRIMO NUMERO
        LD      DE,NUMB2    ;PARTI DAI BIT LSB DEL SECONDO NUMERO
        AND     A
ADDWD:  LD      A,(DE)      ;PRENDI 8 BIT DEL SECONDO NUMERO
        ADC     A,(HL)      ;SOMMA 8 BIT DEL PRIMO NUMERO
        LD      (HL),A      ;MEMORIZZA IL RISULTATO NEL PRIMO NUMERO
        INC     DE
        INC     HL
        DJNZ    ADDWD
        HALT
```

DOMANDE PER I COMMENTI

Secondo, cercate se qualche istruzione non ha funzioni ovvie e segnatela. Qui lo scopo di AND A è di azzerare il Carry all'inizio. Terzo, chiedetevi se i commenti vi dicono ciò che vorreste conoscere se voleste usare il programma, per esempio:

1) Dove è stato inserito il programma? Esistono punti d'ingresso alternativi?
2) Quali parametri sono necessari? Come e in quale forma possono essere dati?
3) Che operazioni realizza il programma?
4) Da dove prende i dati?
5) Dove immagazzina i risultati?
6) Quali casi speciali considera?
7) Che cosa fa il programma riguardo agli errori?
8) Dove esce?

Alcune domande possono non essere attinenti ad un particolare programma e alcune risposte possono essere ovvie. Assicuratevi che non dovreste sedervi e sezionare il programma per calcolare le risposte. Ricordatevi che troppe spiegazioni sono come un bosco estinto che dovrete ripulire dal vostro cammino. C'è qualche cosa che vorreste aggiungere e/o togliere da questo elenco? Se è così, andate avanti — siete uno che ha la sensazione che i commenti debbono essere adeguati e ragionevoli.

```
;ADDIZIONE IN MULTI-PRECISIONE
;
;QUESTO PROGRAMMA REALIZZA UNA ADDIZIONE MULTI-BYTE
;
;INGRESSI:    LOCAZIONE 30H = LUNGHEZZA DEI NUMERI (IN BYTE)
;             LOCAZIONI 41H-50H = PRIMO ADDENDO IN ORDINE LSB→MSB
;             LOCAZIONI 51H-60H = SECONDO ADDENDO
;USCITE:      LOCAZIONI 41H-51H = SOMMA
;
LENGHT  EQU     30H         ;LUNGHEZZA DEI NUMERI
NUMB1   EQU     41H         ;BIT LSB DEL PRIMO NUMERO E RISULTATO
NUMB2   EQU     51H         ;BIT LSB DEL SECONDO NUMERO
        LDA     LENGTH      ;COUNT = LUNGHEZZA DEI NUMERI (IN BYTE)
        LD      B,A
        LD      HL,NUMB1    ;PARTI DAI BIT LSB DEL PRIMO NUMERO
        LD      DE,NUMB2    ;PARTI DAI BIT LSB DEL SECONDO NUMERO
        AND     A           ;AZZERA IL CARRY PER PARTIRE
ADDWD:  LD      A,(DE)      ;PRENDI 8 BIT DEL SECONDO NUMERO
        ADC     A,(HL)      ;SOMMA 8 BIT DEL PRIMO NUMERO
        LD      (HL),A      ;MEMORIZZA IL RISULTATO NEL PRIMO NUMERO
        INC     DE
        INC     HL
        DJNZ    ADDWD
        HALT
```

Esempio 2 di come fare commenti: Uscita di Telescrivente

Il programma basilare è:

```
        LD      A,(60H)
        ADD     A,A
        LD      B,11
TBIT    OUT     (PIODRB),A
        RRA
        SCF
        CALL    B,NBITS
        DJNZ    TBIT
        HALT
```

Commentando i punti importanti ed aggiungendo nomi si ottiene:

```
;PROGRAMMA DI USCITA DELLA TELESCRIVENTE
;
;QUESTO PROGRAMMA STAMPA IL CONTENUTO DELLA LOCAZIONE DI
;   MEMORIA 60H SULLA TELESCRIVENTE
;
;INGRESSI:    LOCAZIONE 60H = CODICE DEL CARATTERE
;USCITE:      NESSUNA
;
TTYPIO  EQU     PIODRB
NBITS   EQU     11              ;NUMERO DI BIT PER CARATTERE
TDATA   EQU     60H             ;INDIRIZZO DEL CARATTERE DA TRASMETTERE
        LD      A,(TDATA)       ;PRENDI IL DATO
        ADD     A,A             ;FAI SCORRERE A SINISTRA
                                ;   E FORMA IL BIT DI PARTENZA
        LD      B,NBITS         ;COUNT = NUMERO DI BIT PER CARATTERE
TBIT:   OUT     (TTYPIO),A      ;INVIA IL BIT ALLA TTY
        RRA                     ;AGGIORNA PER IL PROSSIMO BIT
        SCF                     ;FORMA IL BIT DI STOP (UNO LOGICO)
        CALL    BITDLY          ;RITARDA IL TEMPO DI UN BIT
        DJNZ    TBIT
        HALT
```

Notate come si possa cambiare facilmente questo programma in maniera tale che si possa trasferire una stringa intera di dati, e cominciare dall'indirizzo contenuto nelle locazioni DPTR e DPTR + 1 e finendo con un carattere «03» (ETX in ASCII). Inoltre supponiamo che il terminale sia un dispositivo a 30 caratteri/secondo con un bit di stop (dovremo cambiare la subroutine BITDLY). Cercate di segnare i cambiamenti prima di guardare l'elenco.

```
;PROGRAMMA DI USCITA DI STRINGA
;
;QUESTO PROGRAMMA FA USCIRE UNA STRINGA VERSO IL TERMINALE.
;   LA TRASMISSIONE FINISCE QUANDO SI INCONTRA UN EXT IN CODICE ASCII (30H)
;
;INGRESSI:     LE LOCAZIONI 60H-61H CONTENGONO L'INDIRIZZO DELLA
;                STRINGA CHE DEVE ESSERE FATTA USCIRE
;USCITE:       NESSUNA
;
DPTR    EQU     60H             ;LOCAZIONE DELL'INDIRIZZO DI PARTENZA
                                ;   DEL BUFFER DI USCITA
ENDCH   EQU     03              ;CARATTERE DI FINE = EXT IN ASCII
NBITS   EQU     11              ;NUMERO DI BIT PER CARATTERE
TTYPIO  EQU     PIODRB
        LD      HL,(DPTR)       ;PRENDI L'INDIRIZZO DI PARTENZA DELLA STRINGA
TCHAR:  LD      A,(HL)          ;PRENDI UN CARATTERE
        CP      ENDHC           ;È UN CARATTERE DI FINE?
        JR      Z,DONE          ;SÌ, VAI A DONE
        ADD     A,A             ;FAI SCORRERE IL DATO A SINISTRA
                                ;   E FORMA IL BIT DI PARTENZA
        LD      B,NBITS         ;COUNT = NUMERO DI BIT PER CARATTERE
TBIT:   OUT     (TTYPIO),A      ;INVIA IL BIT ALLA TTY
        RRA                     ;AGGIORNA PER IL PROSSIMO BIT
        SCF                     ;FORMA IL BIT DI STOP (UNO LOGICO)
        CALL    BITDLY          ;RITARDA IL TEMPO DI UN BIT
        DJNZ    TBIT
        INC     HL
        JR      TCHAR
DONE:   HALT
```

Buoni commenti possono semplificare il cambiamento di un programma per soddisfare dei nuovi requisiti. Per esempio, cercate di cambiare l'ultimo programma in modo che esso:

- Faccia iniziare ogni messaggio con il codice ASCII STX (o 2 esadecimale) seguito da un codice di identificazione a 3 digit memorizzato nelle locazioni di memoria da 0030 a 0032.
- Non aggiunga nessun bit di start o di stop.
- Aspetti 1 ms tra due bit.
- Trasmetta 40 caratteri, a partire da quello collocato all'indirizzo DPTR e DPTR + 1.
- Faccia finire ogni messaggio con due caratteri ASCII EXT consecutivi (03 esadecimale).

DIAGRAMMI DI FLUSSO COME DOCUMENTAZIONE

SUGGERIMENTI SULL'USO DEI DIAGRAMMI DI FLUSSO

Nel Capitolo 13 abbiamo già descritto l'uso dei diagrammi di flusso come strumento di progettazione. I diagrammi di flusso sono utili anche nella documentazione, in particolare se:

- Non sono così dettagliati da essere illeggibili.
- I loro punti di decisione siano spiegati chiaramente e segnati.
- Comprendono tutti i salti.
- Corrispondono ai listing reali del programma.
- I diagrammi di flusso sono utili se danno una visione completa del programma. Non sono utili se sono così difficili da leggere come un listing ordinario.

LINGUAGGI A PROGRAMMAZIONE STRUTTURATA COME DOCUMENTAZIONE

Un linguaggio a programmazione strutturata può servire come strumento di documentazione per un programma in linguaggio assembly se:

- Descrivete lo scopo di ogni sezione nei commenti.
- Chiarite quali statement sono inclusi in ogni struttura condizionale o/e loop usando indicatori di inizio e di fine.
- Fate la struttura totale il più semplice possibile.
- Usate un linguaggio efficace e ben definito.

Il programma strutturato vi può aiutare a controllare la logica o/e migliorarla. Inoltre, poiché il programma strutturato è indipendente dalla macchina, vi può aiutare anche nello svolgere lo stesso compito su un altro computer.

MAPPE DI MEMORIA

Una mappa di memoria è una semplice lista di tutte le assegnazioni in un programma. La mappa permette di determinare la quantità di memorie necessarie, le locazioni dei dati o dei sottoprogrammi, e le parti di memoria non allocate. La mappa è un riferimento pratico per trovare locazioni di immagazzinamento e punti di ingresso e per dividere la memoria tra programmatori o routine differenti. La mappa semplificherà pure l'accesso ai dati ed alle subroutine se ne avrete bisogno in ulteriori estensioni o in manutenzione. Qualche volta una mappa grafica è più utile di un listing.

Una mappa tipica potrebbe essere così:

Memoria Programmi		
Indirizzo	Routine	Scopo
0000-0002	RESET	TRASFERISCE IL CONTROLLO AL PROGRAMMA PRINCIPALE NELLA LOCAZIONE 40H
0038-003A	INTRPT	TRASFERISCE IL CONTROLLO AL SERVIZIO DELL'INTERRUPT NELLA LOCAZIONE 300H
0040-0265	MAIN	PROGRAMMA PRINCIPALE
0270-027F	DELAY	PROGRAMMA DEL RITARDO
0280-0290	DSPLY	PROGRAMMA DEL CONTROLLO DEI DISPLAY
0300-0340	KEYIN	PROGRAMMA DI CONTROLLO DELL'INTERRUPT PER LA TASTIERA
Memoria Dati		
1000	NKEYS	NUMERO DI TASTI
1001-1002	KPTR	PUNTATORE AL BUFFER DI TASTIERA
1003-1041	KBFR	BUFFER DI TASTIERA
1042-1051	DBFR	BUFFER DEI DISPLAY
1052-105F	TEMP	MEMORIZZAZIONE TEMPORANEA
10E0-10FF	STACK	STACK DI RAM

La mappa può pure elencare punti d'ingresso addizionali e comprendere una descrizione specifica delle parti di memoria non utilizzate.

LISTE DI PARAMETRI E DI DEFINIZIONI

Liste di parametri e di definizioni all'inizio del programma e di ogni subroutine rendono il programma più semplice da copiare e da cambiare. Le regole seguenti possono essere utili:

REGOLE PER LISTE DI DEFINIZIONI

1) **Separare le locazioni di RAM, le unità di I/O, i parametri, le definizioni, e le costanti del sistema di memoria.**
2) **Preparare liste in ordine alfabetico quando è possibile,** con una descrizione di ogni ingresso.
3) **Dare un nome ad ogni parametro che può cambiare ed includerlo nelle liste.** Tali parametri possono comprendere costanti di tempo, ingressi o codici corrispondenti e particolari chiavi o funzioni, percorsi di controllo o di maschera, caratteri d'inizio o di fine, soglie, ecc.
4) **Mettere le costanti del sistema di memoria in una lista separata.** Queste costanti comprenderanno il RESET e gli indirizzi di servizio degli interrupt, gli indirizzi di partenza del programma, le aree di RAM, le aree di Stack, ecc.
5) **Dare un nome a ogni porta usata da un dispositivo di I/O,** anche se dei dispositivi possono condividere porte nel sistema corrente. La separazione renderà più semplice un'espansione o una riconfigurazione.

Una tipica lista di definizioni può essere:

TIPICA LISTA DI DEFINIZIONI

```
;
;COSTANTI DEL SISTEMA DI MEMORIA
;
RESET   EQU     0       ;INDIRIZZO DI RESET
INTRP   EQU     38H     ;INGRESSO DELL'INTERRUPT
START   EQU     40H     ;INIZIO DEL PROGRAMMA PRINCIPALE
KEYIN   EQU     300H    ;PROGRAMMA DELL'INTERRUPT DA TASTIERA
RAMST   EQU     1000H   ;INIZIO DELLA MEMORIZZAZIONE DEI DATI
STKPTR  EQU     1100H   ;INIZIO DELLO STACK
;
;UNITÀ DI I/O
;
DSPLY   EQU     0E0H    ;USCITA DEL PIO PER I DISPLAY
KBDIN   EQU     0E1H    ;INGRESSO DEL PIO PER LA TASTIERA
KBDOUT  EQU     0E0H    ;USCITA DEL PIO PER LA TASTIERA
TTYPIO  EQU     0F0H    ;PORTA DEL DATO DI TTY
;
;LOCAZIONI DI RAM
;
        ORG     RAMST
NKEYS   DEFS    1       ;NUMERO DI TASTI
KBDPTR  DEFS    2       ;PUNTATORE DEL BUFFER DELLA TASTIERA
KBDBFR  DEFS    40H     ;BUFFER DI INGRESSO DELLA TASTIERA
DSPBFR  DEFS    10H     ;BUFFER DEI DATI DEI DISPLAY
TEMP    DEFS    14H     ;IMMAGAZZINAMENTO TEMPORANEO
;
;PARAMETRI
;
BOUNCE  EQU     2       ;TEMPO DI ELIMINAZIONE DEI RIMBALZI IN MS
GOKEY   EQU     10      ;IDENTIFICAZIONE DEL TASTO 'GO'
MSCNT   EQU     133     ;CONTA UN RITARDO DI 1 MS
OPEN    EQU     0FH     ;CONFIGURAZIONE PER TASTI APERTI
TPULS   EQU     1       ;LUNGHEZZA DELL'IMPULSO PER I DISPLAY IN MS
;
;DEFINIZIONI
;
ALL1    EQU     0FFH    ;CONFIGURAZIONE TUTTI UNO
STCON   EQU     80H     ;DAI L'AVVIO ALL'IMPULSO DI CONVERSIONE
```

Naturalmente gli ingressi delle RAM non saranno abitualmente in ordine alfabetico, poiché il progettista deve metterli in ordine tale da minimizzare il numero dei cambiamenti di indirizzo richiesti nel programma.

ROUTINE DI LIBRERIA

Una documentazione standard di sottoprogrammi vi permetterà di costruire una libreria di programmi utili. L'idea è di rendere questi programmi di facile accessibilità. Un formato standard permetterà a voi o a chiunque altro di vedere che cosa fa il programma. La procedura migliore è di realizzare una forma tipo e di usarla in maniera massiccia. Conservare questi programmi in una forma bene equipaggiata (per esempio in accordo col processore, con il linguaggio e col tipo di programma) ed avrete subito un utile insieme. Ma **ricordatevi che, senza un'organizzazione ed una documentazione appropriata, l'uso delle librerie può essere più difficile che riscrivere il programma da zero.** Fare il debugging di un sistema richiede una precisa comprensione di tutti gli effetti di ogni subroutine.

FORME DI LIBRERIE DI PROGRAMMI STANDARD

Tra le informazioni che avrete bisogno nella forma standard si ha:

- Scopo del programma.
- Processore usato.
- Linguaggio usato.
- Parametri richiesti e come si trasferiscono alle subroutine.
- Risultati ottenuti e come si trasferiscono al programma principale.
- Numero di byte di memoria usati.
- Numero di cicli di clock richiesti. Questo numero può essere un valore medio o tipico o può variare grandemente. Il tempo di esecuzione dipenderà naturalmente dalla velocità del clock del processore.
- Registri interessati.
- Flag interessati.
- Esempio tipico.
- Gestione degli errori.
- Casi speciali.
- Listing del programma documentato.

Se il programma è complesso, la forma delle librerie standard comprenderà pure un diagramma di flusso generale oppure una descrizione del programma in un linguaggio di programmazione strutturata ad alto livello. Come è stato ricordato prima, un programma di libreria sarà molto utile se esso realizza una singola e distinta funzione in maniera ragionevolmente generale.

ESEMPI DI LIBRERIA

Esempio 1 di libreria: Somma di Dati

Scopo: Il programma SUM8 calcola la somma di un insieme di numeri binari ad 8 bit senza segno.

Linguaggio: Assembler Z80.

Condizioni Iniziali: L'indirizzo di partenza dell'insieme di numeri è nella coppia di registri HL, la lunghezza dell'insieme è nell'Accumulatore.

Condizioni Finali: La somma è nell'Accumulatore.

Requisiti:

Memoria — 7 byte.
Tempo — 13 + 26N cicli di clock, dove N è la lunghezza dell'insieme di numeri.
Registri — A, B, H, L.
Tutti i flag sono interessati.

Caso tipico: (tutti i dati in esadecimale)

```
Partenza:
      HL  =  0050
       A  =  03
  (0050)  =  27
  (0051)  =  3E
  (0052)  =  26
    Fine:
       A  =  8B
```

Manipolazione degli errori: Il programma ignora tutti i riporti. Il bit di riporto si riflette solo sull'ultima operazione. Il contenuto iniziale dell'Accumulatore deve essere 1 o maggiore.

Listing:

```
;
;SOMMA DI UN DATO AD 8 BIT
;
SUM8:   LD      B,A         ;COUNT = LUNGHEZZA DEL BLOCCO DI DATI
        SUB     A           ;SOMMA = ZERO
ADD8:   ADD     A,(HL)      ;SOMMA = SOMMA + INGRESSO DEL DATO
        INC     HL
        DJNZ    ADD8
        RET
```

Esempio 2 di Libreria: Conversione decimale sette-segmenti

Scopo: Il programma SEVEN converte un numero decimale in un codice per display a sette-segmenti.

Linguaggio: Assembler Z80.

Condizioni Iniziali: Dati nell'Accumulatore.

Condizioni Finali: Codice a sette-segmenti nell'Accumulatore.

Requisiti:

Memoria — 26 byte, compresa la tabelle a sette segmenti (10 ingressi).
Tempo — 74 cicli di clock se il dato è valido, 40 se non è valido.
Registri — A, B, D, E, H, L.
Tutti i flag sono interessati.

Il dato d'ingresso nell'Accumulatore è distrutto.

Caso tipico: (dato in esadecimale)

```
Partenza:
       A  =  05
    Fine:
       A  =  66
```

Manipolazione degli errori: Il programma rimette zero nell'Accumulatore se il dato non è un digit decimale.

Listing:

```
;
;CONVERSIONE DA DECIMALE A SETTE SEGMENTI
;
SEVEN:  LD      B,0         ;PRENDI IL CODICE DI ERRORE
                            ;   PER SPEGNERE I DISPLAY
        CP      10          ;IL DATO È UN DIGIT DECIMALE?
        JR      NC,DONE     ;NO, CONSERVA IL CODICE DI ERRORE
        LD      L,A         ;SÌ, MUTA IL DATO IN UN INDICE A 16 BIT
        LD      H,0
        LD      DE,SSEG     ;PRENDI L'INDIRIZZO DI BASE DELLA TABELLA
                            ;   A 7 SEGMENTI
        ADD     HL,DE       ;TROVA L'ELEMENTO MEDIANTE INDICIZZAZIONE
        LD      B,(HL)      ;PRENDI IL CODICE A 7 SEGMENTI DALLA TABELLA
DONE:   LD      A,B         ;SALVA IL CODICE A 7 SEGMENTI
                            ;   O IL CODICE DI ERRORE
        RET
SSEG:   DEFB    3FH
        DEFB    06H
        DEFB    5BH
        DEFB    4FH
        DEFB    66H
        DEFB    6DH
        DEFB    7DH
        DEFB    07H
        DEFB    7FH
        DEFB    6FH
```

Esempio 3 di Libreria: Somma decimale

Scopo: Il programma DECSUM addiziona due numeri decimali.

Linguaggio: Assembler Z80.

Condizioni iniziali: L'indirizzo dei bit meno significativi di un numero è nella coppia di registri HL, l'indirizzo dei bit meno significativi dell'altro numero è nella coppia di registri DE, la lunghezza dei numeri (in byte) è in A. I numeri sono disposti a partire dai bit meno significativi negli indirizzi inferiori.

Condizioni finali: La somma sostituisce il numero con l'indirizzo di partenza nella coppia di registri HL.

Requisiti:

Memoria — 11 byte.
Tempo — 13 + 50N cicli di clock, dove N è il numero dei byte coinvolti.
Registri — A, B, D, E, H, L.
Tutti i flag sono interessati. Il Carry mostra se la somma produce un riporto.

Caso tipico: (dato in esadecimale)

```
Partenza:
      HL = 0060
      DE = 0050
       A = 2
  (0060) = 34
  (0061) = 55
  (0050) = 88
  (0051) = 15
    Fine:
  (0060) = 22
  (0061) = 71
   CARRY = 0
```

Manipolazione degli errori: Il programma non controlla la validità degli ingressi decimali. L'Accumulatore deve essere 1 o maggiore.

Listing:

```
DECSUM: LD      B,A        ;CONTEGGIO = LUNGHEZZA DEI NUMERI (IN BYTE)
        AND     A          ;AZZERA IL CARRY PER INIZIARE
DECADD: LD      A,(DE)     ;PRENDI 2 DIGIT DECIMALI DALLA STRINGA 2
        ADC     A,(HL)     ;AGGIUNGERE UNA COPPIA DI DIGIT
                           ;   DALLA STRINGA 1
        DAA                ;FAI L'ADDIZIONE DECIMALE
        LD      (HL),A     ;IMMAGAZZINA IL RISULTATO NELLA STRINGA 1
        INC     DE
        INC     HL
        DJNZ    DECADD
        RET
```

DOCUMENTAZIONE TOTALE

STRUTTURA DELLA DOCUMENTAZIONE

Una documentazione completa del software di un microprocessore comprenderà tutti o la gran parte degli elementi ricordati. In tale modo **una struttura di una documentazione totale può comprendere:**

- **Diagrammi di flusso generali.**
- **Una descrizione scritta del programma.**
- **Una lista di tutti i parametri e le definizioni.**
- **Una mappa di memoria.**
- **Un listing documentato del programma.**
- **Una descrizione del piano di collaudo e dei risultati di test.**

La documentazione può comprendere anche:

- **Diagrammi di flusso dei programmatori.**
- **Diagrammi di flusso dei dati.**
- **Programmi strutturati.**

Le procedure di documentazione sottolineate sopra sono il minimo insieme di documenti accettabile per un software di tipo non produttivo. Un software adatto alla produzione richiede sforzi di documentazione anche maggiori. Si potrebbero realizzare anche i documenti seguenti:

- Manuale Logico di Programma.
- Guida per l'Utilizzatore.
- Manuale di Manutenzione.

Il manuale logico del programma si sviluppa sulla spiegazione scritta realizzata col software. Dovrebbe essere scritto per un individuo tecnicamente competente che può anche non possedere la conoscenza particolareggiata supposta nella spiegazione scritta nel software. Il manuale logico di programma dovrebbe spiegare quali sono gli scopi di progetto del sistema, quali algoritmi sono stati scelti per realizzare questi scopi e quali prestazioni si devono ottenere nel raggiungerli.

Dovrebbe spiegare in grande dettaglio quali strutture di dati sono state impiegate e come siano state manipolate. Dovrebbe fornire una guida passo-passo delle elaborazioni interne del codice. Infine, dovrebbe contenere qualsiasi tabella o grafico speciali che aiutino a spiegare qualunque concetto racchiuso nel codice. Dovrebbe comprendere pure carte di conversione di codici, diagrammi di stato, matrici di traduzione e diagrammi di flusso.

La guida dell'utilizzatore è probabilmente la più importante e la più esaminata parte della documentazione. Non importa come un sistema sia stato progettato bene in quanto esso sarebbe inutile se nessuno potrà usarlo efficientemente. La guida dell'utilizzatore dovrebbe fornire a tutti gli utenti, sofisticati o no, una introduzione al sistema. Dovrebbe quindi fornire una spiegazione dettagliata delle caratteristiche di sistema e del loro uso.

Usate molti esempi perchè un buon esempio può cristallizzare le informazioni contenute in molte pagine di testo. Dovrebbero essere date direttive passo-passo. Verificate la guida dell'utilizzatore, cioè sottoponete a dura prova le procedure di utilizzazione passo-passo quando le avete documentate. I programmatori con conoscenza particolareggiata di un progetto di sistema spesso prendono scorciatoie non del tutto evidenti al generico lettore. Si potrebbe scrivere un libro intero su come scrivere guide per utilizzatori, ma una ulteriore trattazione va al di là dello scopo di questo libro. Tuttavia, ricordatevi che non vi sforzerete mai troppo nel preparare una guida per l'utilizzatore, poiché essa sarà la più usata di tutti i documenti di sistema.

Il manuale di manutenzione è progettato per il programmatore che deve modificare il sistema. Dovrebbe mettere in evidenza le procedure passo-passo per quelle riconfigurazioni progettate nel sistema. Inoltre dovrebbe evidenziare ogni provvedimento relativo al codice per una espansione futura.

La documentazione non dovrebbe essere presa alla leggera o posticipata alla fine dello sviluppo del software. Una documentazione appropriata, insieme con una adeguata esperienza di programmazione, non è solo una parte importante del prodotto finale ma può pure rendere lo sviluppo più semplice, più veloce e più produttivo. Il progettista dovrebbe rendere coerente e completa la parte di documentazione di ogni stadio di sviluppo del software.

RIPROGETTAZIONE

Qualche volta il progettista può trovarsi nella necessità di spremere la velocità fino all'ultimo microsecondo o di comprimere la memoria fino all'ultimo byte in un programma. Da quando si sono rese disponibili memorie a singolo chip di maggiore capacità, il problema della memoria è diventato meno rilevante. Il problema del tempo naturalmente, è rilevante solo se l'applicazione è critica nei confronti del tempo; in molte applicazioni il microprocessore spende gran parte del suo tempo in attesa di dispositivi esterni e la velocità del programma non è il fattore di maggior importanza.

COSTO DI RIPROGETTAZIONE

Il comprimere un programma fino all'ultimo bit è raramente così importante come alcuni scrittori vorrebbero far credere. Anzitutto, l'esecuzione risulta costosa per i seguenti motivi:

1) Necessita di tempo supplementare per il programmatore che è spesso il costo singolo maggiore nello sviluppo del software.
2) Sacrifica la struttura e la semplicità con conseguente aumento nel tempo di collaudo e di debugging.
3) I programmi necessitano di un'ulteriore documentazione.
4) I programmi che ne risultano saranno più difficoltosi da ampliare, da manutenere o da riutilizzare.

Secondariamente il minore costo unitario e la maggior prestazione può essere non veramente importante. Un costo minore ed una maggiore prestazione potranno veramente incrementare le vendite? Oppure non sarebbe meglio offrire caratteristiche più orientate all'utilizzatore? **Le sole applicazioni che sembrano giustificare sforzi e tempo supplementari sono quelle di volume molto alto, di basso costo e a basse prestazioni, per le quali il costo di un ulteriore chip di memoria sarà trascurabile rispetto al costo dello sviluppo software extra.** Per altre applicazioni vi accorgerete di stare giocando una partita costosa senza ragione.

RIORGANIZZAZIONE DI MAGGIORE O DI MINORE IMPORTANZA

Tuttavia, se dovete riprogettare un programma, i seguenti suggerimenti vi saranno utili. Determinate, dapprima, quali prestazioni massime e quale minor uso di memoria sono necessari. Se si richiede un miglioramento del 25% o minore, potete ottenerlo riorganizzando il programma. Se è maggiore del 25%, avete fatto un errore di progetto fondamentale; sarà necessario che consideriate drastici mutamenti nell'hardware o nel software. Tratteremo dapprima della riorganizzazione e successivamente dei mutamenti drastici. Dovreste pure vedere il Capitolo 5 di Z80 Programmazione per il Progetto Logico per alcuni esempi.

Si può notare in particolare che il risparmio di memoria può essere critico se deve permettere di mettere un programma nella limitata quantità di ROM e di RAM disponibile in un semplice microcomputer ad uno o due chip. Il costo dell'hardware per sistemi piccoli può in tal modo essere ridotto in modo sostanziale, se i loro requisiti possono essere limitati alla dimensione della memoria e alle limitazioni di I/O di quel particolare sistema ad uno o due chip.

RIORGANIZZAZIONE PER USARE MENO MEMORIA

Le seguenti procedure ridurranno il consumo di memoria per programmi in linguaggio assembly delle Z80:

RISPARMIO DI MEMORIA

1) **Sostituire codici in linee ripetitive con subroutine.** Assicuratevi, tuttavia, che le istruzioni CALL e RETURN non coprano gran parte del guadagno. È da notare che questa sostituzione usualmente produce programmi più lenti a causa del tempo speso nel trasferire il controllo avanti ed indietro.

2) **Usare operazioni su registri dove possibile.** Ma ricordatevi del costo della inizializzazione supplementare.

3) **Usare lo Stack dove possibile.** Lo Stack Pointer è aggiornato automaticamente dopo ogni utilizzazione in modo tale che non sono necessarie esplicite istruzioni di aggiornamento.

4) **Eliminare le istruzioni di Jump.** Cercate di organizzare il programma o di usare istruzioni di salti indiretti (JP(HL) o JP (IX o IY)), RST o RETURN.

5) **Sfruttare i vantaggi da indirizzi che si possono gestire come quantità ad 8 bit.** Questi comprendono la pagina zero e indirizzi multipli di 100 in esadecimale. Per esempio, dovreste cercare di posizionare tutte le tabelle di ROM in una sola sezione di 100_{16} byte di memoria e tutte le variabili in un'altra sezione di 100_{16} byte.

6) **Organizzare i dati e le tabelle in modo tale che si possano indirizzare senza preoccuparsi dei riporti dovuti al calcolo dell'indirizzo o senza alcuna effettiva indicizzazione.** Ciò vi permetterà ancora di gestire indirizzi a 16 bit come quantità ad 8 bit. Vedere le pagine da 5-1 a 5-6 di Z80 Programmazione per il Progetto Logico per degli esempi.

7) **Usare istruzioni a 16 bit per sostituire due separate operazioni ad 8 bit.** Ciò può essere particolarmente utile in fase di inizializzazione o di immagazzinamento dei risultati.

8) **Usare risultati intermedi di precedenti sezioni del programma.**

9) **Sfruttare i vantaggi di istruzioni** quali INC (HL), DCR (HL), LD(HL), RL(HL) e RR(HL), **che agiscono direttamente su locazioni di memoria senza usare registri.**

10) **Usare INC e DEC per posizionare ad 1 o/e 0 i bit dei flag.**

11) **Usare salti relativi piuttosto che salti ad indirizzamento diretto.**

12) **Sfruttare i vantaggi delle istruzioni di Spostamento di Blocco, di Ricerca di Blocco e di I/O di blocco** ogni volta che dovete gestire blocchi di dati.

13) **Prestare attenzione alle speciali e brevi forme di istruzioni** quali gli spostamenti nell'Accumulatore (RLCA, RLA, RRCA e RRA) e DJNZ.

14) **Usare algoritmi piuttosto che tabelle** per calcolare espressioni aritmetiche o logiche e per effettuare conversioni di codici. È da notare che questa sostituzione può produrre programmi più lenti.

15) **Ridurre la dimensione di tabelle matematiche con interpolazioni** tra gli ingressi. Nuovamente, qui si risparmia memoria a scapito del tempo di esecuzione.

16) **Sfruttare il vantaggio dell'insieme di registri alternativi per ridurre l'uso di memorizzazione.** Questo può far risparmiare anche tempo.

Sebbene alcuni metodi di riduzione dell'impiego di memorie risparmino anche tempo, si può in generale risparmiare una quantità di tempo apprezzabile solo prendendo in considerazione i loop eseguiti frequentemente.

RISPARMIO DI TEMPO DI ESECUZIONE

Anche eliminando completamente un'istruzione eseguita una sola volta si possono risparmiare al più pochi microsecondi. Ma un risparmio in un loop eseguito frequentemente sarà moltiplicato molte più volte.

Quindi, **se dovete ridurre il tempo di esecuzione, procedete come segue:**

1) **Determinare quale frequenza di esecuzione ha ogni loop del programma.** Potete fare ciò manualmente o usando un simulatore del software o un altro metodo di verifica.

2) **Esaminare i loop nell'ordine determinato dalla loro frequenza di esecuzione,** a partire dalla più frequente. Continuare a rivedere l'elenco fino a raggiungere la riduzione richiesta.

3) **Anzitutto, vedere se esistono operazioni che possono essere spostate al di fuori del loop,** per esempio, calcoli ripetitivi, dati che possono essere messi in un registro o nello Stack, indirizzi che possono essere messi in coppie di registri o in registri indice, casi speciali o errori che possono essere gestiti altrove, ecc. Si noti che questo richiederà una inizializzazione e memoria supplementari, ma risparmierà tempo.

4) **Cercare di eliminare statement di tipo Jump.** Questi utilizzano molto tempo. Oppure, usare salti con indirizzamento diretto che richiedono più memoria ma minor tempo dei salti con indirizzamento relativo.

5) **Sostituire subroutine con codice in linea.** Ciò risparmierà almeno una istruzione CALL ed una istruzione RETURN.

6) **Usare lo Stack per immagazzinamento temporaneo di dati.**

7) **L'uso di ogni suggerimento menzionato per risparmiare memoria diminuirà pure il tempo di esecuzione.** Questi suggerimenti comprendono istruzioni di gestione di blocchi, indirizzi ad 8 bit, istruzioni a 16 bit, RST, forme speciali e brevi di istruzioni, ecc.

8) **Inoltre non guardate le istruzioni eseguite una sola volta.** Cambiamenti effettuati su tali istruzioni possono creare occasioni per errori senza alcun guadagno apprezzabile.

9) **Evitare indirizzamenti indicizzati e relativi ogni qualvolta sia possibile,** in quanto essi impiegano tempo in più.

10) **Usare tabelle piuttosto che algoritmi;** fare le tabelle maneggevoli il più possibile anche se devono ripetere molti ingressi.

MAGGIORE RIORGANIZZAZIONE

Se necessita un incremento superiore al 25% nella velocità o una riduzione nell'utilizzo della memoria, non cercate di riorganizzare il codice. Le possibilità che avete di ricavare un netto miglioramento sono scarse a meno che non richiediate la presenza di un esperto dall'esterno. Sarebbe preferibile realizzare una modifica di importanza più rilevante.

ALGORITMI MIGLIORI

La variazione più ovvia è quella di un algoritmo migliore. Se si stanno eseguendo in particolare classificazioni, ricerche, o calcoli matematici, si può scoprire un metodo più veloce o più breve nella letteratura. Le librerie di algoritmi sono disponibili in alcuni giornali e da gruppi professionali. Vedi, per esempio, i Riferimenti Bibliografici da 1 a 10 al termine di questo capitolo.

Più hardware può sostituire del software. I contatori, i registri a scorrimento, le unità aritmetiche, i moltiplicatori hardware, e altri veloci dispositivi possono comportare risparmio sia di tempo che di memoria. I calcolatori, gli UART, le tastiere, i codificatori ed altri dispositivi più lenti possono comportare risparmio di memoria anche se funzionano in modo lento. Le interfacce seriali e parallele compatibili, ed altri dispositivi designati specificatamente all'utilizzo con lo Z80 possono far risparmiare tempo, sgravando la CPU di alcuni compiti ad essa assegnati.

Altre variazioni possono inoltre risultare utili:

ALTRE VARIAZIONI DI MAGGIORE IMPORTANZA

1) **Una CPU con una parola più lunga risulterà più veloce** se il dato è sufficientemente lungo. Una tale CPU utilizzerà meno memoria globalmente. I processori a 16 bit, ad esempio, impiegano la memoria in maniera assai più efficiente dei processori ad 8 bit, dato che la maggior parte delle loro istruzioni hanno una lunghezza pari ad una parola.

2) **Possono esistere versioni della CPU che funzionano a velocità di clock più elevate.** Ma ricordate che ci sarà la necessità di utilizzare della memoria e delle porte di I/O più veloci, e si dovranno adattare tutti i cicli di ritardo.

3) **Due CPU possono essere in grado di eseguire il loro compito funzionando in parallelo** oppure separatamente se possono dividersi i compiti e risolvere i problemi relativi alla trasmissione reciproca delle informazioni.

4) **Uno speciale processore microprogrammato può essere in grado di eseguire lo stesso programma assai più velocemente.** Tuttavia, il costo risulterà molto maggiore anche se si utilizza un'emulazione fuori banco.

5) **Si possono fare dei compromessi tra tempo e memoria.** Le tabelle di riferimento e le ROM di funzione risulteranno più veloci degli algoritmi, ma occuperanno più memoria.

DECISIONE SU UNA VARIAZIONE DI GRANDE RILEVANZA

Questo tipo di problema, nel quale è necessario un notevole perfezionamento, solitamente è conseguenza della mancanza di un'adeguata programmazione degli stadi di definizione e di progetto. Nello stadio di definizione del problema si dovrebbe determinare quale processore e quali metodi saranno adatti alla trattazione del problema. Se si compiono degli errori di valutazione, successivamente il costo sarà alto. Una soluzione conveniente può comportare un dispendio ingiustificato di tempo di sviluppo. Non cercate proprio di procedere oltre; la migliore soluzione è solitamente quella di eseguire il progetto in modo preciso e di contrassegnare gli errori che servono come esperienza. Se si sono eseguiti metodi quali i diagrammi di flusso, la programmazione modulare, la programmazione strutturata, la progettazione di tipo «top-down» e la documentazione adatta, si sarà in grado di risparmiare una grande quantità di lavoro anche se si deve eseguire una variazione di più rilevante importanza.

BIBLIOGRAFIA

1. Collected Algorithms form ACM. ACM, Inc. P.O. Box 12105. Church Street Station, New York 10249.

2. Chen, T. C., "Automatic Computation of Exponentials, Logarithms, Ratios, and Square Roots", IBM Journal of Research and Development. Volume 18, pp. 380-388, Luglio 1972.

3. H. Schmid, Decimal Computation, Wiley-Interscience, New York, 1974.

4. Knuth, D. E., The Art of Computer Programming, Volume 1: Fundamental Algorithms, Addison-Wesley, Reading, Mass., 1967.

5. Knuth, D.E., The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Addison - Wesley, Reading, Mass., 1969.

6. Knuth, D. E., The Art of Computer Programming, Volume 3: Sorting and Searching, Addison-Wesley, Reading, Mass., 1973.

7. Carnahan, B. et al., Applied Numerical Methods, Wiley, New York, 1969.

8. Despain, A. M. "Fourier Transform Computers Using CORDIC Iterations", IEEE Transactions on Computers, Ottobre 1974, pp. 993-1001.

9. Luke, Y. L., Algorithms for the Computation of Mathematical Functions, Academic Press, New York, 1977.

10. Hwang, K., Computer Arithmetic, Wiley, New York, 1978.

11. Dollhoff, T., "Microprocessor Software: How to Optimize Timing and Memory Usage. Part Four: Techniques fort the Zilog Z80". Digital Design, Febbraio 1977, pp. 44-51.

Capitolo 16

PROGETTI CAMPIONE

PROGETTO # 1: Un Cronometro Digitale

PROCEDURA D'INGRESSO DEL CRONOMETRO

Scopo: Questo progetto consiste in un cronometro digitale. L'operatore scrive due cifre (minuti e decimi di minuti) da una tastiera simile a quelle per computer e poi preme il tasto GO. Il sistema conta all'indietro il tempo rimanente su due visualizzatori LED a sette segmenti (vedi il Capitolo 11 per una descrizione di tastiere non codificate e di visualizzatori LED).

Hardware: Il progetto impiega una porta d'ingresso ed una porta di uscita (un Dispositivo Z80 Ingresso/Uscita Parallelo chiamato PIO), due visualizzatori a sette segmenti, una tastiera con 12 tasti, un circuito invertitore 7404, e/o una porta NAND 7400 a seconda della polarità dei visualizzatori a sette segmenti. I visualizzatori possono necessitare di driver, di invertitori e di resistenze, a seconda della loro polarità e configurazione.

L'hardware è organizzato come indicato in Figura 16-1. Le linee d'uscita 0, 1 e 2 sono usate per esplorare la tastiera. Le linee d'ingresso 0, 1, 2 e 3 sono usate per determinare se qualche tasto è stato premuto. Le linee di uscita 0, 1, 2 e 3 sono usate per inviare le cifre BCD ai decodificatori/driver a sette segmenti. La linea di uscita 4 è usata per attivare i visualizzatori LED (se la linea 4 è '1', i visualizzatori sono accesi). La linea di uscita 5 è usata per selezionare il visualizzatore sinistro oppure destro; la linea di uscita 5 è '1' se si deve utilizzare il visualizzatore di sinistra, '0' se quello da utilizzare è il visualizzatore di destra. Così, la linea comune sul visualizzatore sinistro dovrebbe risultare attiva se la linea 4 e la linea 5 sono '1', mentre la linea comune sul visualizzatore di destra dovrebbe risultare attiva se la linea 4 è '1' e la linea 5 è '0'. La linea di uscita 6 controlla il punto decimale situato a destra del visualizzatore di sinistra. Questa può essere pilotata da un circuito invertitore o semplicemente lasciata nella condizione «on».

Collegamenti della Tastiera: La Tastiera è una semplice tastiera da computer disponibile per 50 ¢ da un fornitore del luogo. Essa consiste di 12 interruttori a pulsante non codificati disposti in quattro righe di tre colonne ciascuna. Dato che il collegamento della tastiera non coincide con le righe e le colonne osservate, il programma fa uso di una tabella per individuare i tasti. Le Tabelle 16-1 e 16-12 contengono le connessioni d'ingresso e uscita per la tastiera. Il tasto del punto decimale è presente a vantaggio dell'operatore e per una possibile espansione futura; il programma in corso in realtà non utilizza il tasto.

In una applicazione realistica, la tastiera richiederebbe la presenza di resistenze di pullup per garantire che gli ingressi vengano realmente letti come '1' logici quando i tasti non vengono attivati. Sarebbero inoltre necessarie delle resistenze di limitazione della corrente oppure diodi sulla porta di uscita per evitare di danneggiare i driver nel caso in cui due uscite risultassero attive contemporaneamente. Ciò potrebbe verificarsi se fossero premuti nello stesso momento due tasti della stessa riga, collegando in tal modo due diverse uscite di colonna.

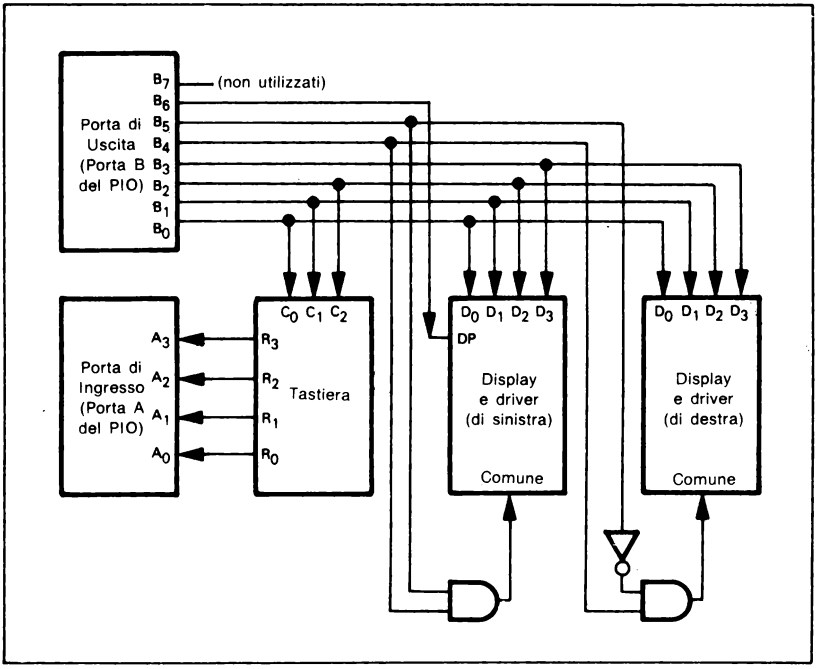


Figura 16-1. Configurazione di I/O di un Cronometro Digitale

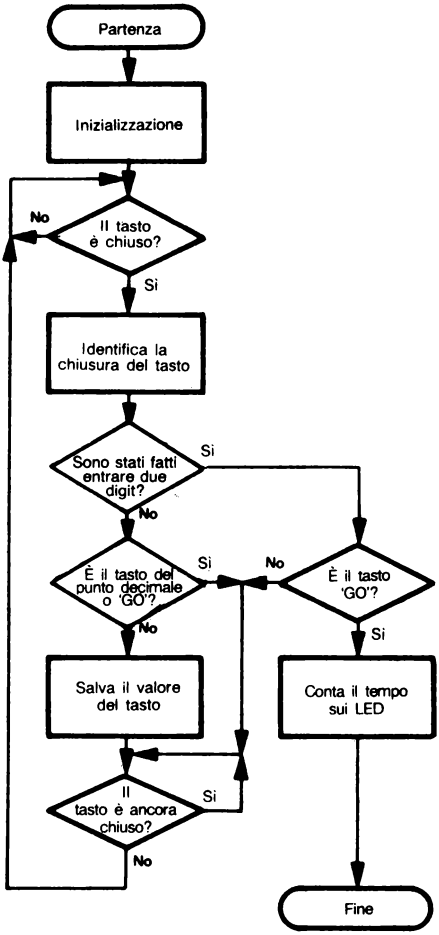
Tabella 16-1. Collegamenti d'ingresso per la Tastiera del Cronometro

Bit di Ingresso	Tasti Collegati
0	'3', '5', '8'
1	'2', '6', '9'
2	'0', '1', '7'
3	'4', '.', 'GO'

Tabella 16-2. Collegamenti d'uscita per la Tastiera del Cronometro

Bit di Uscita	Tasti Collegati
0	'0', '2', '3', '4'
1	'1', '8', '9', 'GO'
2	'5', '6', '7', '.'

Diagramma di Flusso del Programma Generale:



Collegamenti dai Visualizzatori: I visualizzatori sono del tipo a sette segmenti e posseggono internamente i propri decodificatori integrali. Un tipico esempio potrebbe essere il dispositivo della Texas Instrument TIL 309, che possiede un chip interno TTL MSI con latch, decodificatore e driver. Chiaramente i visualizzatori standard a sette segmenti sarebbero più economici ma richiederebbero del software supplementare (la routine di conversione a sette segmenti mostrata nel Capitolo 7). Il dato viene inviato al visualizzatore sotto forma di singola cifra decimale codificata in binario; le cifre sono rappresentate come indica la Figura 11-15. Il punto decimale è un singolo LED che viene acceso quando l'ingresso relativo al punto decimale è un '1' logico. Si possono ottenere ulteriori chiarimenti per quanto riguarda i visualizzatori nei Riferimenti Bibliografici 12 e 13 riportati al termine di questo capitolo.

Descrizione del Programma:

Il programma è modulare ed ha diverse subroutine. I punti da mettere in rilievo sono la chiarezza e la generalità piuttosto che l'efficienza; ovviamente, il programma non utilizza le intere possibilità del processore Z80. Ciascuna sezione del listing verrà ora descritta dettagliatamente.

1) **Commenti Introduttivi:**
I commenti introduttivi descrivono completamente il programma; questi commenti costituiscono un riferimento affinché altri utilizzatori possano facilmente applicare, estendere e comprendere lo stesso programma. I formati, i capoversi e le spaziature standard rendono più semplice la leggibilità del programma.

2) **Definizioni delle Variabili:**
Tutte le definizioni delle variabili vengono poste all'inizio del programma in modo tale che possano essere facilmente controllate e modificate. Ciascuna variabile viene posta in un elenco alfabetico con le altre variabili dello stesso tipo; i commenti descrivono il significato di ogni variabile. Le categorie sono:

a) Costanti del sistema di memoria che possono variare da sistema a sistema a seconda dello spazio di memoria assegnato a diversi programmi o tipi di memorie.
b) Memorizzazioni temporanee (RAM) usate come variabili.
c) Indirizzi di porte di I/O (PIO).
d) Definizioni.

Le costanti del sistema di memoria sono poste nelle definizioni in modo che l'utilizzatore possa riallocare il programma, le memorizzazioni temporanee, e lo stack di memoria senza effettuare qualsiasi altra variazione. Le costanti di memoria possono essere modificate per adattare altri programmi o per farle coincidere con una assegnazione di indirizzi ROM e RAM di un particolare sistema.

Le memorizzazioni temporanee sono assegnate per mezzo delle psuedo-operazioni DEFS («Define Storage»). Una pseudo-operazione ORG (origine) colloca le locazioni di memorizzazione temporanea in una particolare zona della memoria. In queste locazioni non viene posto alcun valore in modo tale che il programma potrebbe eventualmente essere collocato in ROM o PROM e il sistema potrebbe essere attivato dal «reset» iniziale d'alimentazione senza necessità di ricaricamento. A ciascun indirizzo di porta impegnato da un PIO viene assegnato un nome cosicché si possono facilmente mutare gli indirizzi per manipolare le configurazioni modificate. La designazione serve anche a distinguere in modo netto i registri di controllo dai registri dei dati.

Le definizioni chiarificano il significato di certe costanti e consentono la modifica dei parametri in modo semplice. Ogni definizione viene fornita nella forma (binaria, esadecimale, ottale, ASCII, o decimale) nella quale risulta più chiaro il suo significato. Alcuni parametri (come il tempo di eliminazione del rimbalzo) vengono assegnati nel nostro caso in modo da poter essere modificati secondo le necessità del sistema.

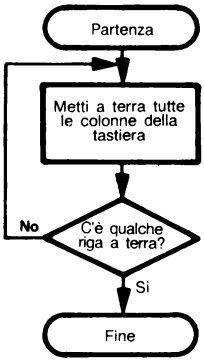
3) **Inizializzazione**

La locazione di memoria 0 (la locazione di «reset» nel microprocessore Z80) contiene un salto all'indirizzo di partenza del programma principale. Quest'ultimo può essere pertanto collocato in qualsiasi zona di memoria ed essere raggiunto tramite un segnale di «RESET». L'inizializzazione consiste di quattro fasi:

a) Collocare un valore di partenza nel Puntatore di Stack. Lo Stack viene impiegato solamente per memorizzare indirizzi di ritorno da subroutine.
b) Programmare i registri di controllo del PIO.
c) Inizializzare a zero il numero dei tasti di cifra pigiati.
d) Inizializzare la locazione dove il prossimo tasto di cifra premuto verrà salvato all'inizio della matrice dei tasti di cifra. Viene impiegata una procedura indiretta, nella quale KEYAD contiene l'indirizzo nel quale verrà collocata la prossima cifra. Ogni volta che viene riconosciuto un tasto di cifra, il contenuto di KEYAD viene decrementato in modo che il successivo tasto di cifra verrà posto nella successiva locazione di memoria.

4) **Attesa dell'Attivazione di un Tasto**

Diagramma di Flusso:



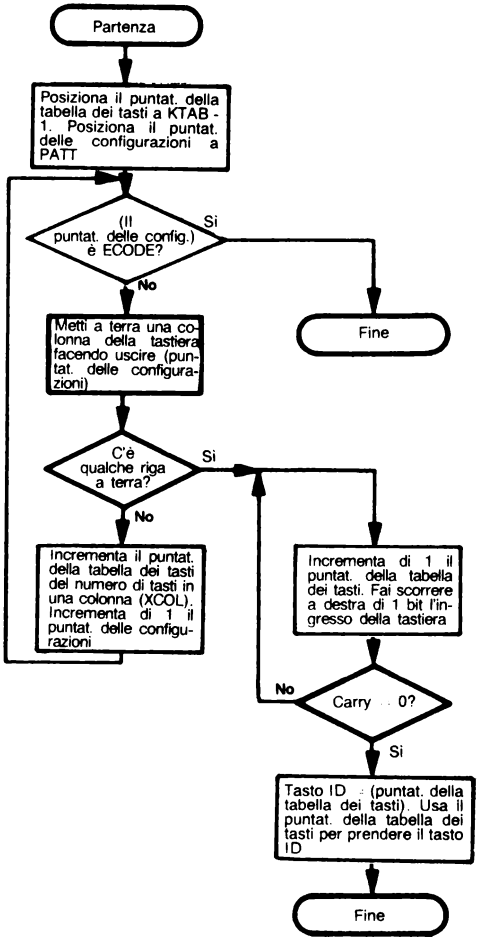
Le attivazioni dei tasti sono identificate ponendo al potenziale di terra tutte le colonne della tastiera e controllando quindi se vi sono delle righe allo stesso potenziale (cioè, le chiusure dei contatti riga-colonna). Notate che il programma non presume che i bit degli ingressi inutilizzati siano tutti a livello alto; invece, i bit inerenti alla tastiera vengono mascherati mediante un'istruzione logica AND.

5) **Tasto senza Rimbalzo**

Il programma non considera i rimbalzi presenti alla chiusura del tasto mediante un espediente software attendendo, cioè, due millisecondi. Questo è di solito un tempo sufficientemente lungo per un contatto pulito. Il sottoprogramma DELAY conta semplicemente 1 millisecondo mediante il registro C. Il numero di millisecondi è contenuto nell'Accumulatore. DELAY dovrebbe essere adattato se s'intendono utilizzare memorie più lente o un clock più lento. Si potrebbe eseguire la variazione semplicemente ridefinendo la costante MSCNT.

6) **Identificazione dell'Attivazione di un Tasto**

Diagramma di Flusso:



Il particolare tasto premuto viene identificato ponendo al potenziale di terra le singole colonne ed osservando se viene individuata una chiusura. Una volta che accade ciò (così la colonna dei tasti è nota), la riga dei tasti può essere determinata spostando l'ingresso.

I formati richiesti per porre a terra le singole colonne della tastiera sono presenti in una tabella PATT in memoria. Il formato finale nella tabella è un contrassegno (ECODE) che indica che tutte le colonne sono state collegate a terra senza che sia stata individuata una chiusura. Questo formato indica inoltre al programma principale che non può essere identificata la chiusura (ad esempio la chiusura terminata di un tasto o di un errore hardware sono avvenuti prima che si potesse individuare la chiusura).

TABELLA DEI TASTI

Le identificazioni dei tasti sono contenute nella tabella KTAB in memoria. I tasti della prima colonna (collegati al bit di uscita meno significativo) sono seguiti da quelli della seconda colonna, ecc. All'interno di una colonna, il tasto della riga collegata al bit d'ingresso meno significativo è il primo, ecc. Così, ogni volta che viene esplorata una colonna senza trovare una chiusura, il numero di tasti di una colonna (NROWS) deve essere aggiunto al puntatore di tabella dei tasti onde spostarsi verso la successiva colonna. Il puntatore di tabella dei tasti viene inoltre decrementato di un'unità prima che venga esaminato ciascun bit negli ingressi di riga; questo processo si ferma quando viene individuato un ingresso pari a zero. Notate che il puntatore di tabella dei tasti viene inizializzato una locazione prima della tabella, dato che esso viene sempre incrementato una volta nella ricerca della particolare riga.
Se non si può identificare la chiusura di un tasto, non lo si prende semplicemente in considerazione e ci si aspetta un'altra chiusura.

7) **Azione nel Caso d'Identificazione di un Tasto**
Se il programma possiede sufficienti cifre (due in questo semplice caso), esso si attende solamente l'attivazione del tasto GO e trascura tutti gli altri tasti. Se trova un tasto di cifra, esso ne conserva il valore nella matrice dei tasti, incrementa il numero di tasti di cifra presenti e incrementa anche il puntatore della matrice dei tasti.

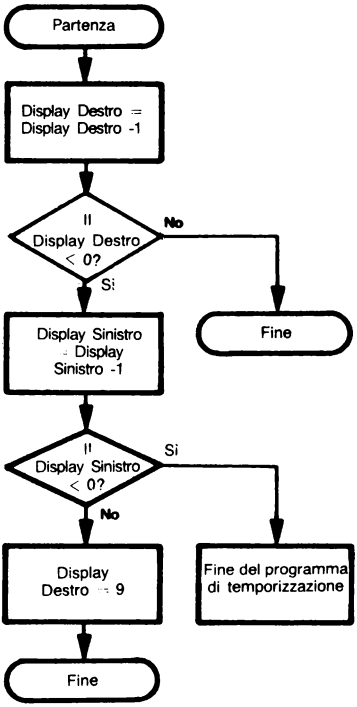
Se l'ingresso non è completo, il programma deve attendere la chiusura di un tasto per terminare in modo che il sistema non leggerà di nuovo la stessa chiusura. L'utilizzatore deve attendere tra una chiusura di tasti e l'altra (vale a dire, rilasciare un tasto prima di premerne un altro). Notate che il programma identificherà le doppie chiusure di tasti come un tasto oppure l'altro, a seconda di quale chiusura individua per prima la routine di identificazione. Una versione perfezionata di questo programma potrebbe visualizzare le cifre come vengono immesse all'ingresso e permetterebbe all'utilizzatore di tralasciare uno zero non significativo (cioè, premere i tasti «.», «7», «GO» per ottenere un conteggio pari a sette decimi di minuto).

8) **Sistemazione delle Uscite del Visualizzatore**
Le cifre vengono poste nei registri o nelle locazioni di memoria con il bit 4 al valore «1» in modo che l'uscita venga inviata ai visualizzatori. I bit 5 e 6 sono posizionati ad «1» per la cifra più significativa per dirigere l'uscita al visualizzatore di sinistra e per accendere il punto decimale.

9) **Pulsazione dei Visualizzatori LED**
Ciascun visualizzatore viene acceso per due millisecondi. Questo processo viene ripetuto 1500 volte onde ottenere un ritardo complessivo di 0,1 minuti, o 6 secondi. Gli impulsi sono sufficientemente frequenti così da far apparire accesi di luce continua i visualizzatori LED.

10) **Decremento del Conteggio del Visualizzatore**

Diagramma di Flusso:



Il valore della cifra meno significativa viene ridotto di uno. Se ciò influenza il bit 4 (LEDON — usato per accendere i visualizzatori), la cifra è diventata negativa. Un prestito deve essere pertanto ottenuto dalla cifra più significativa. Se il prestito dalla cifra più significativa influenza il bit 4, il conteggio è giunto oltre lo zero e il conteggio alla rovescia è terminato. Altrimenti, il programma posiziona il valore della cifra meno significativa a 9 e continua.

Notate che i commenti descrivono sia le sezioni del programma che i singoli statement. I commenti spiegano cosa sta facendo il programma, non cosa fanno gli specifici codici operativi. La spaziatura e l'identificazione sono state visualizzate per migliorare la leggibilità.

```
;NOME DEL PROGRAMMA: TIMER
;DATA DEL PROGRAMMA: 24/10/78
;PROGRAMMATORE: LANCE A. LEVENTHAL
;REQUISITI DEL PROGRAMMA: D1 (209) BYTE
;REQUISITI DI RAM: 5 BYTE
;REQUISITI DI I/O: PORTA D'INGRESSO 1, PORTA D'USCITA 1 (PIO 1 DELLO Z80)
;
;
;QUESTO PROGRAMMA È UN TEMPORIZZATORE IN SOFTWARE CHE ACCETTA
;   INGRESSI DA UNA TASTIERA TIPO CALCOLATORE E QUINDI
;   FORNISCE IL CONTEGGIO ALLA ROVESCIA DI UN CRONOMETRO SU DUE
;   DISPLAY LED A 7 SEGMENTI IN MINUTI E IN DECIMI DI MINUTI
;
;TASTIERA
;
;SI È PRESUPPOSTA UNA TASTIERA A 12 TASTI
;TRE COLLEGAMENTI DI COLONNA SONO FATTI USCIRE DALL'ELABORATORE
;   IN MODO CHE UNA COLONNA DEI TASTI PUÒ ESSERE MESSA A TERRA
;QUATTRO RIGHE SONO COLLEGATE AD INGRESSI DELL'ELABORATORE IN
;   MODO DA IDENTIFICARE I CIRCUITI REALIZZATI
;LA TASTIERA È LIBERATA DAI RIMBALZI ASPETTANDO DUE MILLISECONDI
;   DOPO AVER RICONOSCIUTO UNA CHIUSURA DI UN TASTO
;UNA NUOVA CHIUSURA DI UN TASTO VIENE IDENTIFICATA ATTENDENDO
;   LA FINE DELLA PRECEDENTE POICHÈ NON VIENE USATO NESSUN
;   SEGNALE DI STROBE
;LE COLONNE DELLA TASTIERA SONO COLLEGATE AI BIT DA 0 A 2
;   DELLA PORTA B DEL PIO
;LE RIGHE DELLA TASTIERA SONO COLLEGATE AI BIT DA 0 A 3
;   DELLA PORTA A DEL PIO
;
;DISPLAY
;
;SI USANO DUE DISPLAY LED A SETTE SEGMENTI CON DECODIFICATORI
;   SEPARATI (7447 OPPURE 7448 IN FUNZIONE DEL TIPO DI DISPLAY)
;I DATI DI INGRESSO DEL DECODIFICATORE SONO COLLEGATI AI BIT DA
;   0 A 3 DELLA PORTA B DEL PIO
;IL BIT 4 DELLA PORTA B DEL PIO SI USA PER ATTIVARE I DISPLAY
;   A LED (IL BIT 4 È 1 PER INVIARE DATI AI LED)
;IL BIT 5 DELLA PORTA B DEL PIO È USATO PER SCEGLIERE QUALE
;   LED DEVE ESSERE USATO (IL BIT 5 È 1 SE SI DEVE USARE IL
;   PRIMO DISPLAY, O SE SI DEVE USARE IL SECONDO DISPLAY)
;IL BIT 6 DELLA PORTA B DEL PIO PER ACCENDERE IL PUNTO DECIMALE
;   SUL PRIMO DISPLAY (IL BIT 6 È 1 SE IL DISPLAY DEVE ESSERE ILLUMINATO)
;
;METODO
;
;PASSO 1 — INIZIALIZZAZIONE
;   IL PUNTATORE DI STACK DELLA MEMORIA (USATO PER GLI INDIRIZZI
;   DI RITORNO DEI SOTTOPROGRAMMI) VIENE INIZIALIZZATO. IL
;   NUMERO DI TASTI DI CIFRA PREMUTI È POSTO A ZERO, E L'INDIRIZZO
;   DOVE VERRÀ POSTA L'IDENTIFICAZIONE DEL SUCCESSIVO TASTO
;   VIENE INIZIALIZZATO AL PRIMO INDIRIZZO NELL'INSIEME DEI TASTI
;   DI CIFRA
;PASSO 2 — ATTESA DI UNA CHIUSURA DI UN TASTO
;   TUTTE LE COLONNE DELLA TASTIERA SONO MESSE A TERRA E LE RIGHE
;   DELLA TASTIERA VENGONO ESAMINATE FINO A CHE NON SI TROVA
;   UN CIRCUITO CHIUSO
```

```
;PASSO 3 – ELIMINAZIONE DEI RIMBALZI DELLA CHIUSURA DI UN TASTO
;   SI INTRODUCE UN'ATTESA DI 2 MILLISECONDI PER ELIMINARE
;   I RIMBALZI DEL TASTO
;PASSO 4 – IDENTIFICAZIONE DELLA CHIUSURA DI UN TASTO
;   LA CHIUSURA DI UN TASTO È IDENTIFICATA METTENDO A TERRA UNA
;   SOLA COLONNA E DETERMINANDO LA RIGA E LA COLONNA DELLA
;   CHIUSURA DEL TASTO. SI USA UNA TABELLA PER CODIFICARE I TASTI
;   SECONDO IL LORO NUMERO DI RIGA E DI COLONNA.
;   NELLA TABELLA DEI TASTI, I DIGIT SONO IDENTIFICATI DAL LORO
;   VALORE, IL TASTO DEL PUNTO DECIMALE È IL NO. 10, E IL
;   TASTO «GO» È IL NUMERO 11
;PASSO 5 – SALVATAGGIO DELLA CHIUSURA DI UN TASTO
;   LE CHIUSURE DEI TASTI DI DIGIT SONO SALVATE NELL'ARRAY DEI
;   TASTI DI DIGIT FINCHÈ NON SI IDENTIFICANO DUE DIGIT. PUNTI
;   DECIMALI, ULTERIORI DIGIT, E CHIUSURE DEL TASTO «GO» PRIMA
;   CHE SIANO IDENTIFICATI DUE DIGIT VENGONO IGNORATI
;   DOPO AVER TROVATO DUE DIGIT, SI USA IL TASTO «GO» PER FAR
;   PARTIRE IL PROCESSO DI CONTO ALLA ROVESCIA
;PASSO 6 – INTERVALLO DI TEMPORIZZAZIONE DEL CONTO ALLA ROVESCIA
;   SUI LED
;   SI REALIZZA UN CONTO ALLA ROVESCIA SUI LED, AVENDO IL PRIMO
;   DIGIT CHE RAPPRESENTA IL NUMERO RESTANTE DI MINUTI E IL SECONDO
;   DIGIT CHE RAPPRESENTA IL NUMERO RESTANTE DELLE DECINE DI MINUTI
;
;
;
;DEFINIZIONI DELLE VARIABILI DEL TEMPORIZZATORE
;COSTANTI DEL SISTEMA DI MEMORIA
;
BEGIN   EQU     50H             ;BEGIN È LA LOCAZIONE DI PARTENZA PER PROG
LASTM   EQU     1000H           ;LASTM È L'INDIRIZZO DI PARTENZA DELLO STACK
TEMP    EQU     800H            ;TEMP È L'INIZIO DELL'IMMAGAZZINAMENTO IN RAM
;IMMAGAZZINAMENTO TEMPORANEO IN RAM
        ORG     TEMP
KEYAD:  DEFS    2               ;KEYAD CONTIENE L'INDIRIZZO NELL'ARRAY DEI
                                ;   TASTI DI DIGIT NEL QUALE SARÀ POSTA L'IDENTI-
                                ;   FICAZIONE DEL PROSSIMO TASTO DI DIGIT
KEYNO:  DEFS    2               ;KEYNO È L'ARRAY DEI TASTI DI DIGIT – ESSO
                                ;   CONTIENE LE IDENTIFICAZIONI DEI TASTI
                                ;   DI DIGIT CHE SONO STATI PREMUTI
NKEYS:  DEFS    1               ;NKEYS CONTIENE IL NUMERO DEI TASTI
                                ;   DI DIGIT PREMUTI
;
;UNITÀ DI I/O ED INDIRIZZI DEL PIO
;
PIODRA  EQU     0E0H            ;INGRESSO DEL PIO PER LA TASTIERA
PIOCRA  EQU     0E2H
PIODRB  EQU     0E1H            ;USCITA DEL PIO PER LA TASTIERA E PER IL DISPLAY
PIOCRB  EQU     0E3H
;
;DEFINIZIONI
;
DECPT   EQU     6               ;POSIZIONE DEL BIT PER ACCENDERE IL LED
                                ;   DEL PUNTO DECIMALE
ECODE   EQU     0FFH            ;CODICE DI ERRORE SE LA ROUTINE ID
                                ;   NON TROVA IL TASTO
```

```
GOKEY   EQU     11          ;IDENTIFICAZIONE DEL NUMERO PER IL TASTO «GO»
LEDON   EQU     4           ;POSIZIONE DEL BIT PER INVIARE LE USCITE AI LED
LEDSL   EQU     5           ;POSIZIONE DEL BIT PER SCEGLIERE IL 1 DISPLAY
MSCNT   EQU     0F9H        ;CONTEGGIO NECESSARIO PER DARE
                            ;   UN RITARDO DI 1MS
MXKEY   EQU     2           ;NUMERO MASSIMO DI CHIUSURE
                            ;   UTILIZZATE DEI TASTI DI DIGIT
NROWS   EQU     4           ;NUMERO DI RIGHE NELLA TASTIERA
                            ;   O TASTI NELLA COLONNA
OPEN    EQU     00001111B   ;INGRESSO DA TASTIERA SE NON È CHIUSO
                            ;   NESSUN TASTO
TPULS   EQU     2           ;NUMERO DI MS TRA DUE DISPLAY DI DIGIT
TWAIT   EQU     2           ;NUMERO DI MS PER ELIMINARE RIMBALZI DAI TASTI
;
;
;
        ORG     0
;
;ROUTINE DI RESET PER ARRIVARE AL PROGRAMMA DEL TEMPORIZZATORE
;
        JP      BEGIN       ;TROVA IL PROGRAMMA DEL TEMPORIZZATORE
;
;INIZIALIZZAZIONE DEL PROGRAMMA DEL TEMPORIZZATORE
;
        ORG     BEGIN
        LD      A,01001111B ;LA PORTA A DEL PIO DIVENTA UN INGRESSO
        OUT     (PIOCRA),A
        LD      A,00001111B ;LA PORTA B DEL PIO DIVENTA UN'USCITA
        OUT     (PIOCRB),A
        LD      SP,LASTM    ;METTI LO STACK ALLA FINE DELLA MEMORIA
        SUB     A
        LD      (NKEYS),A   ;NUMERO DI TASTI DI DIGIT PREMUTI = ZERO
        LD      HL,KEYNO    ;LOCAZIONE DI PARTENZA PER I TASTI DI DIGIT
        LD      (KEYAD),HL
;
;SCANSIONE DELLA TASTIERA CERCANDO CHIUSURE DI TASTI
;
;START:  CALL    SCANC       ;ATTESA DELLA CHIUSURA DI UN TASTO
;
;ATTESA PER ELIMINARE I RIMBALZI DEL TASTO
;
        LD      A,TWAIT     ;DAI IL TEMPO DI ELIMINAZIONE DI RIMBALZI IN MS
        CALL    DELAY       ;ATTESA DELLA FINE DEI RIMBALZI DI UN TASTO
;
;IDENTIFICARE QUALE TASTO È STATO PREMUTO
;
        CALL    IDKEY       ;IDENTIFICA LA CHIUSURA DEL TASTO
        CP      ECODE       ;È STATA IDENTIFICATA LA CHIUSURA DEL TASTO?
        JR      Z,START     ;NO, ATTESA DI UN'ALTRA CHIUSURA
;
;AZIONE SULL'IDENTIFICAZIONE DEL TASTO
        LD      B,A         ;SALVA IL NUMERO DEL TASTO
        LD      HL,NKEYS    ;SCEGLI IL NUMERO MAGGIORE DEI TASTI DI DIGIT
        LD      A,(HL)
        CP      MXKEY       ;SI È RAGGIUNTO IL MASSIMO?
        JR      Z,KEYF      ;SÌ, RICERCA DEL TASTO GO
```

```
        LD      A,B             ;NO, RICERCA SOLO I TASTI DI DIGIT
        CP      10              ;QUESTO TASTO È UN DIGIT?
        JR      NC,WAITK        ;NO, IGNORALO
        INC     (HL)            ;SÌ, INCREMENTA IL CONTATORE DEI TASTI DI DIGIT
        LD      HL,(KEYAD)      ;SALVA IL NUMERO DEL TASTO NELL'ARRAY
        LD      (HL),A
        INC     HL
        LD      (KEYAD),HL
;
;ATTESA CHE FINISCA LA CHIUSURA DEL TASTO CORRENTE
;
WAITK:  CALL    SCANO           ;ATTESA CHE IL TASTO SIA RILASCIATO
        JR      START           ;ATTESA DELLA CHIUSURA DEL TASTO SUCCESSIVO
;
;RICERCA DEL TASTO GO SE SI SONO TROVATI TASTI A SUFFICIENZA
;
KEYF:   LD      A,B             ;PRENDI IL NUMERO DEI TASTI PREMUTI
        CP      GOKEY           ;È IL TASTO «GO»?
        JR      NZ,WAITK        ;NO, IGNORALO
;
;METTI I DIGIT NEI REGISTRI PER I DISPLAY
;
        LD      HL,KEYNO
        LD      D,(HL)          ;PRENDI IL PRIMO DIGIT
        SET     DECPT,D         ;ACCENDI IL PUNTO DECIMALE
        SET     LEDON,D         ;POSIZIONA LE USCITE DEI LED
        SET     LEDSL,D         ;SCEGLI IL PRIMO DISPLAY
        INC     HL
        LD      E,(HL)          ;PRENDI L'ULTIMO DIGIT
        SET     LEDON,E         ;POSIZIONA LE USCITE DEI LED
;
;FAI PULSARE I DISPLAY A LED
;
        LD      C,PIODRB        ;PRENDI L'INDIRIZZO DELLA PORTA DI USCITA
LEDLP:  LD      H,6             ;POSIZIONA I CONTATORI A 6 SECONDI
TLOOP:  LD      B,250
LDPUL:  OUT     (C),D           ;USCITA DEL PRIMO DIGIT PER IL LED 1
        LD      A,TPULS         ;RITARDO TRA DUE DIGIT
        CALL    DELAY
        OUT     (C),E           ;USCITA DELL'ULTIMO DIGIT PER IL LED 2
        LD      A,TPULS         ;RITARDO TRA DUE DIGIT
        CALL    DELAY
        DJNZ    LDPUL
        DEC     H
        JR      NZ,TLOOP
;
;DECREMENTA IL CONTEGGIO DEI DISPLAY A LED
        DEC     E               ;CONTEGGIO ALLA ROVESCIA DELL'ULTIMO DIGIT
        BIT     LEDON,E         ;L'ULTIMO DIGIT PASSATO È ZERO?
        JR      NZ,LEDLP        ;NO, CONTINUA
        DEC     D               ;CONTEGGIO ALLA ROVESCIA DEL PRIMO DIGIT
        BIT     LEDON,D         ;IL PRIMO DIGIT PASSATO È ZERO?
        JP      Z,BEGIN         ;SÌ, ATTESA DEL PROSSIMO COMPITO
                                ;   DI TEMPORIZZAZIONE
        LD      E,9             ;NO, POSIZIONA A 9 L'ULTIMO DIGIT
        SET     LEDON,E         ;POSIZIONA L'USCITA PER I LED
        JR      LEDLP           ;RITORNA ALLA SEZIONE DEI DISPLAY
```

```
;
;SUBROUTINE SCANC CHE SCANDISCE LA TASTIERA IN
;   ATTESA DI UNA CHIUSURA DI UN TASTO. TUTTI GLI INGRESSI
;   DELLA TASTIERA SONO MESSI A TERRA
;
SCANC:  SUB     A               ;MESSA A TERRA DI TUTTE LE COLONNE
                                ;   DELLA TASTIERA
        OUT     (PIODRB),A
        IN      A,(PIODRA)
        AND     OPEN            ;IGNORA GLI INGRESSI NON USATI
        CP      OPEN            ;C'È QUALCHE TASTO CHIUSO?
        JR      Z,SCANC         ;NO, CONTINUA LA SCANSIONE
        RET
;
;SUBROUTINE DELAY CHE ATTENDE IL NUMERO DI MILLISECONDI
;   SPECIFICATO NEL REGISTRO A
;
DELAY:  EXX                     ;SALVA I REGISTRI DELL'UTILIZZATORE
DLY1:   LD      C,MSCNT         ;CARICA IL REGISTRO C PER 1 MS
WTLP:   DEC     C               ;ATTESA DI 1 MS
        JR      NZ,WTLP
        DEC     A               ;CONTEGGIO ALLA ROVESCIA DEL NUMERO DI MS
        JR      NZ,DLY1
        EXX                     ;RIMEMORIZZA I REGISTRI DELL'UTILIZZATORE
        RET
;
;LA SUBROUTINE IDKEY DETERMINA IL NUMERO DI RIGA E COLONNA RELATIVI ALLA
;   CHIUSURA DEL TASTO ED IDENTIFICA IL TASTO STESSO CON L'IMPIEGO
;   DI UNA TABELLA
;
IDKEY:  LD      BC,PATT         ;PUNTA ALLE CONFIGURAZIONI DELLA SCANSIONE
        LD      HL,KTAB-1       ;INIZIALIZZA IL PUNTATORE
                                ;   DELLA TABELLA DEI TASTI
        LD      DE,NROWS        ;PRENDI IL NUMERO DI TASTI IN UNA COLONNA
;
;SCANSIONE SUCCESSIVA DELLE COLONNE DELLA TASTIERA
;   CERCANDO UNA RICHIUSURA
;
FCOL:   LD      A,(BC)          ;PRENDI LA CONFIGURAZIONE PER METTERE
                                ;   A TERRA LA COLONNA
        CP      ECODE           ;SONO STATE ESPLORATE TUTTE LE COLONNE?
        RET     Z               ;SÌ, RITORNA COL CODICE DI ERRORE
        OUT     (PIODRB),A      ;SCANSIONE DELLE COLONNE
        IN      A,(PIODRA)
        AND     OPEN            ;IGNORA GLI INGRESSI NON UTILIZZATI
        CP      OPEN            ;C'È QUALCHE TASTO CHIUSO IN QUESTA COLONNA?
        JR      NZ,FROW         ;SÌ, DETERMINA LA CHIUSURA DELLA RIGA
        ADD     HL,DE           ;NO, SPOSTA IL PUNTATORE DELLA TABELLA
                                ;   DEI TASTI VERSO LA COLONNA SUCCESSIVA
        INC     BC              ;PUNTA ALLA PROSSIMA CONFIGURAZIONE
                                ;   DI SCANSIONE
        JR      FCOL
;
;DETERMINA IL NUMERO DI RIGA DELLA CHIUSURA
;
FROW:   INC     HL              ;SPOSTA IL PUNTATORE DELLA TABELLA DEI TASTI
```

```
                                ;   VERSO LA RIGA SUCCESSIVA
        RRCA                    ;LA RIGA SUCCESSIVA È A TERRA?
        JR      C,FROW          ;NO, CONTINUA LA RICERCA
;
;IDENTIFICA IL TASTO DALLA TABELLA
;
        LD      A,(HL)          ;PRENDI IL NUMERO DEL TASTO
        RET
;
;ESPLORA LE CONFIGURAZIONI USATE PER METTERE A TERRA
;   UNA COLONNA ALLA VOLTA
;CONFIGURAZIONE DI ERRORE USATA PER INDICARE CHE SONO
;   STATE ESPLORATE TUTTE LE COLONNE
;LA COLONNA COLLEGATA AL BIT 0 DI USCITA È ESPLORATA PER
;   PRIMA, QUINDI QUELLA COLLEGATA AL BIT 1 D'USCITA, ETC.
;
PATT:   DEFB    00000110B
        DEFB    00000101B
        DEFB    00000011B
        DEFB    ECODE
;
;TABELLA DELLA TASTIERA
;
;LE COLONNE SONO INDICI PRINCIPALI, LE RIGHE INDICI SECONDARI
;I TASTI NELLA COLONNA COLLEGATA AL BIT 0 DI USCITA SONO
;   SEGUITI DA QUELLI NELLA COLONNA COLLEGATA AL BIT 1 DI
;   USCITA, ETC. IN UNA COLONNA, IL TASTO COLLEGATO AL BIT 0 DI
;   INGRESSO È IL PRIMO SEGUITO DA QUELLO COLLEGATO AL
;   BIT 1 DI INGRESSO, ETC.
;I TASTI DI DIGIT SONO DA 0 A 9, IL PUNTO DECIMALE È 10, GO È 11
;
KTAB:   DEFB    3               ;C0,R0
        DEFB    2               ;C0,R1
        DEFB    0               ;C0,R2
        DEFB    4               ;C0,R3
        DEFB    8               ;C1,R0
        DEFB    9               ;C1,R1
        DEFB    1               ;C1,R2
        DEFB    11              ;C1,R3
        DEFB    5               ;C2,R0
        DEFB    6               ;C2,R1
        DEFB    7               ;C2,R2
        DEFB    10              ;C2,R3
;
;SUBROUTINE SCANO CHE ESPLORA LA TASTIERA IN ATTESA
;   CHE LA CHIUSURA DI UN TASTO TERMINI COSÌ CHE VENGA
;   TROVATA LA CHIUSURA SUCCESSIVA
;
SCANO:  SUB     A               ;METTI A TERRA TUTTE LE COLONNE
                                ;   DELLA TASTIERA
        OUT     (PIODRB),A
        IN      A,(PIODRA)
        AND     OPEN            ;IGNORA GLI INGRESSI NON UTILIZZATI
        CP      OPEN            ;CI SONO ANCORA TASTI CHIUSI?
        JR      NZ,SCANO        ;SÌ, CONTINUA LA SCANSIONE
        RET
        END
```

PROGETTO # 2: Un Termometro Digitale

Scopo: Questo progetto consiste in un termometro digitale che rappresenta la temperatura in gradi Celsius su due visualizzatori a sette segmenti.

Hardware: Il progetto impiega una porta d'ingresso ed una porta d'uscita, due visualizzatori a sette segmenti, un circuito invertitore 74LS04, una porta NAND 74LS00 o una porta AND 74LS08 a seconda della polarità dei visualizzatori, un convertitore A/D monolitico ad 8 bit per Dispositivi Analogici AD 7570J, un comparatore LM 311, e vari driver di periferiche, resistenze e condensatori come richiesto dai visualizzatori e dal convertitore. (Vedi il Capitolo 11 ed il Riferimento Bibliografico 1 al termine di quel capitolo per le discussioni relative ai convertitori A/D).

La Figura 16-2 rappresenta l'organizzazione dell'hardware. La linea d'uscita 7 della porta B del PIO viene impiegata per inviare un segnale di Conversione di Partenza al convertitore A/D. Le linee d'ingresso da 0 a 7 sono collegate direttamente alle otto linee di dati digitali dal convertitore. Le linee di uscita da 0 a 3 vengono utilizzate per inviare delle cifre BCD ai decodificatori/driver a sette segmenti. La linea di uscita 4 attiva i visualizzatori e la linea di uscita 5 seleziona il visualizzatore sinistro o destro (la linea 5 è '1' per il visualizzatore di sinistra).

HARDWARE ANALOGICO DEL TERMOMETRO

La parte analogica dell'hardware è rappresentata in Figura 16-3. Il termistore fornisce semplicemente una resistenza che dipende dalla temperatura. La Figura 16-4 rappresenta l'andamento della resistenza e la Figura 16-5 il campo di variazione dei valori di corrente lungo il quale la resistenza è lineare. La conversione a gradi Celsius nel programma viene eseguita mediante una tabella di calibrazione. I due potenziometri possono essere regolati per graduare i dati in modo appropriato. Viene generato un clock per il convertitore A/D da una rete RC. I valori sono R7 = 33 K Ω e C1 = 1000 pF, cosicché la frequenza di clock è di circa 75KHz. A questa frequenza, il tempo massimo di conversione per otto bit è circa 50 microsecondi. Si permette un ritardo molto più lungo per la conversione in modo che non sia più necessaria alcuna verifica per il termine della conversione stessa. La versione ad 8 bit del convertitore necessita delle seguenti connessioni particolari. Le otto linee dei dati sono costituite da DB2 fino a DB9 (DB1 è sempre a livello alto durante la conversione e DB0 è a livello basso). L'ingresso di Ciclo Breve ad 8 bit (pin 26-$\overline{SC8}$) è collegato al potenziale basso in modo che venga eseguita solamente una conversione ad 8 bit. Nel caso considerato, l'Abilitazione di Byte Alto (pin 20-HBEN) e l'Abilitazione di Byte Basso (pin 21-LBEN) erano ambedue collegate al potenziale alto in modo tale che le uscite dei dati fossero sempre abilitate.

Il convertitore utilizza il metodo delle approssimazioni successive. Esso paragona ciascun bit all'uscita di un convertitore D/A per vedere se quel bit dovrebbe essere attivo oppure no. Ciascun confronto necessita di un periodo di clock. Il metodo è veloce ma soggetto ad errori se l'ingresso contiene del rumore. Dei metodi più accurati per la manipolazione di I/O analogici vengono descritti nei riferimenti bibliografici al termine di questo capitolo.

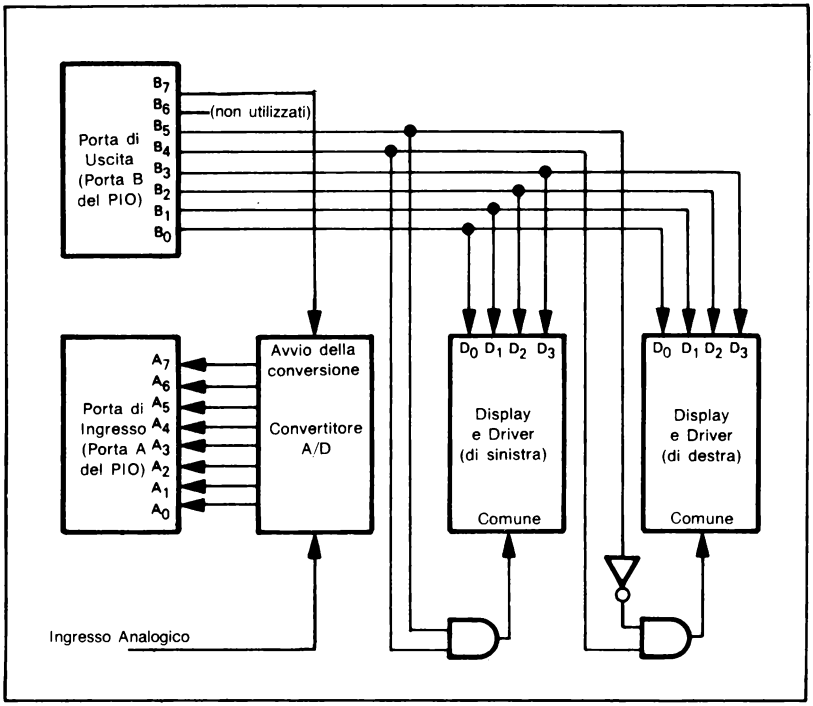
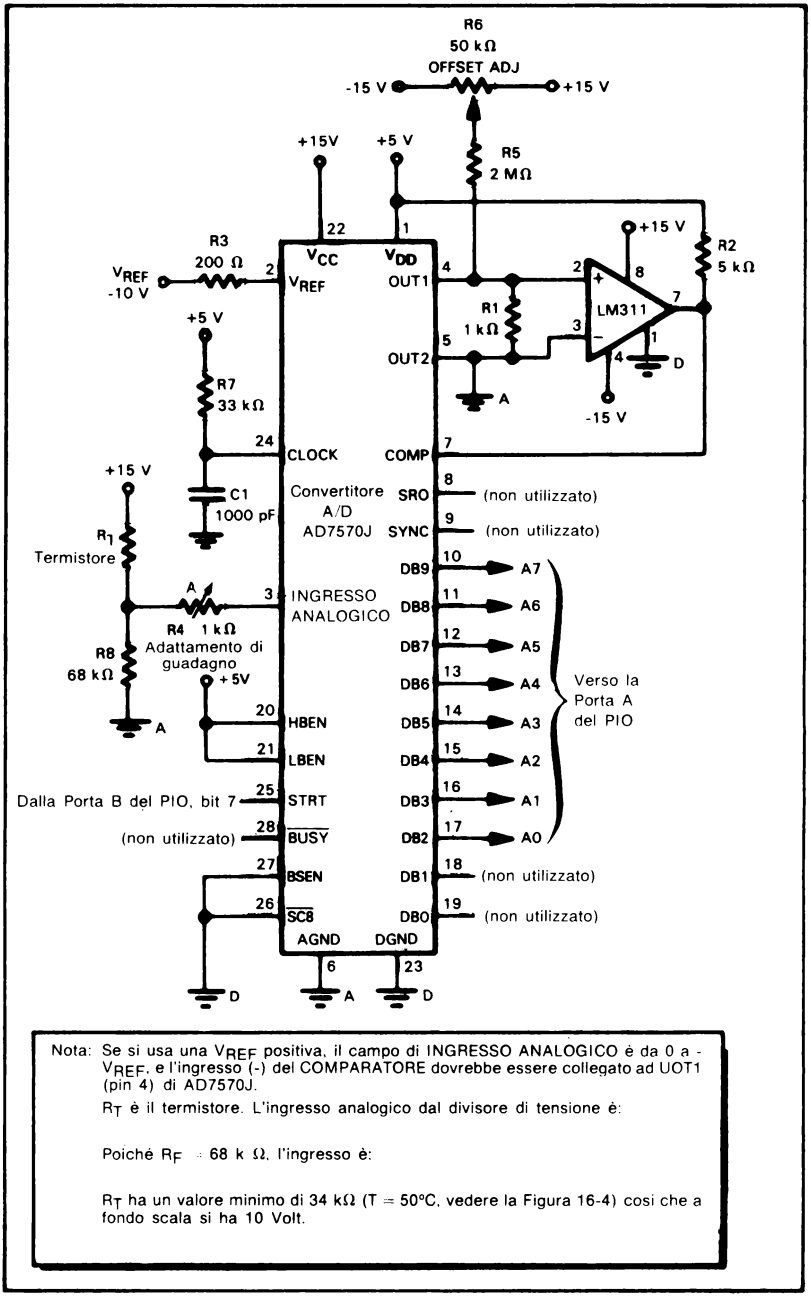


Figura 16-2. Configurazione di I/O di un Termometro Digitale



Nota: Se si usa una V_{REF} positiva, il campo di INGRESSO ANALOGICO è da 0 a -V_{REF}, e l'ingresso (-) del COMPARATORE dovrebbe essere collegato ad UOT1 (pin 4) di AD7570J.

R_T è il termistore. L'ingresso analogico dal divisore di tensione è:

Poiché R_F = 68 k Ω, l'ingresso è:

R_T ha un valore minimo di 34 kΩ (T = 50°C, vedere la Figura 16-4) cosi che a fondo scala si ha 10 Volt.

Figura 16-3. Hardware Relativo al Termometro Digitale

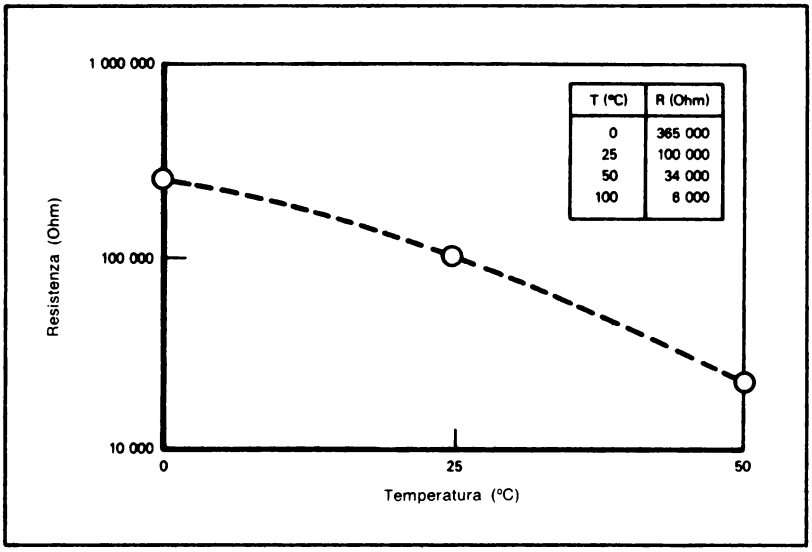


Figura 16-4. Caratteristiche del Termistore (Fenwal GA51J1 Bead)

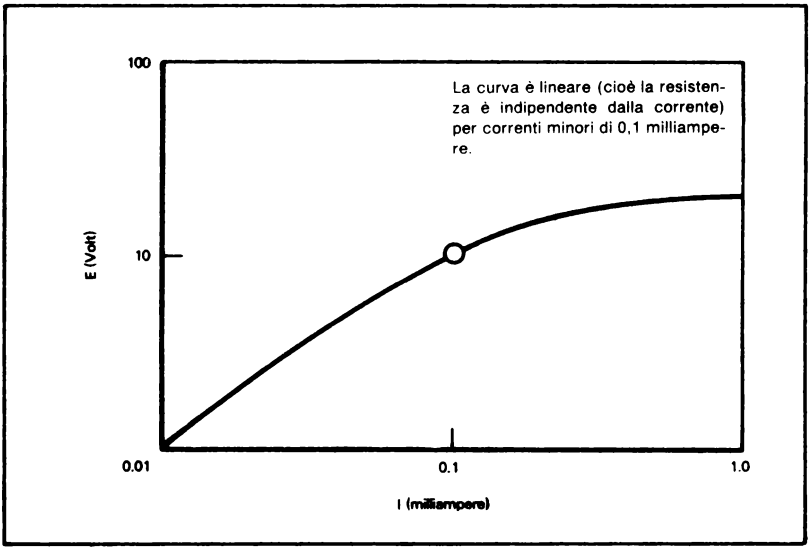
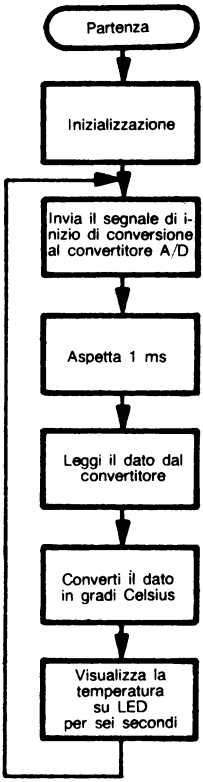


Figura 16-5. Curva E-I Tipica del Termistore (25°C)

Diagramma di Flusso del Programma Generale:



Descrizione del Programma:

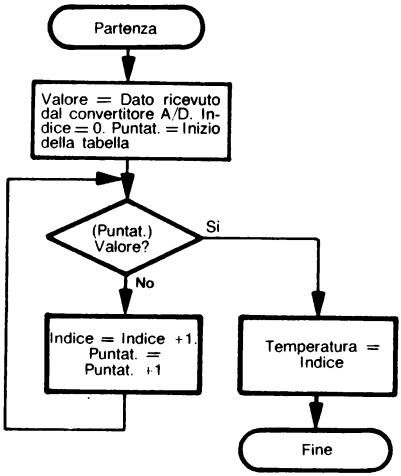
1) **Inizializzazione**
La locazione 0 (la locazione di RESET del microprocessore Z80) contiene un salto all'indirizzo di partenza del programma principale. L'inizializzazione configura i registri di controllo del PIO e inizializza il Puntatore di Stack all'indirizzo più elevato nella RAM. Lo Stack viene impiegato esclusivamente per memorizzare gli indirizzi di ritorno dai sottoprogrammi.

2) **Invio del segnale CONVERSIONE INIZIALE al Convertitore A/D**
La CPU genera un impulso sulla linea CONVERSIONE INIZIALE presentando per prima cosa un «1» sulla linea 7 della porta B del PIO ed in seguito uno «0» su quella stessa linea. Ciascun ingresso proveniente dal convertitore necessita di un impulso iniziale.

3) **Attesa di 1 ms per la Conversione**
Un ritardo di 1 ms dopo l'impulso di CONVERSIONE INIZIALE garantisce il completamento della conversione. In realtà, il convertitore necessita solamente di un massimo di 100 microsecondi per una conversione ad 8 bit. Si potrebbe ridurre il ritardo tenendo sotto controllo il segnale BUSY proveniente dal convertitore. Questo segnale è «1» (conversione completa) oppure «0» (conversione in corso) se viene indirizzata la linea «$\overline{\text{BUSY}}$ ENABLE». Nel caso in esame non v'è alcuna ragione per accelerare il processo di conversione. Chiaramente, si possono utilizzare gli interrupt con il segnale $\overline{\text{BUSY}}$ collegato alla linea «STROBE» del PIO.

4) **Lettura dei Dati dal Convertitore A/D**
La lettura dei dati comporta una semplice operazione di «input». Si deve notare che il convertitore per Dispositivi Analogici AD 7570J possiede un ingresso «Enable» e delle uscite tristate cosicché esso può essere collegato direttamente al Bus Dati del microprocessore. Il convertitore 7570 è, naturalmente, sottoutilizzato in questa speciale applicazione, in particolare poiché lo si sta interfacciando con il microprocessore Z80 attraverso un PIO. Un convertitore A/D ad 8 bit più semplice come il dispositivo National 5357 eseguirebbe lo stesso compito con minor costo; questo dispositivo è disponibile in contenitore a 18 terminali, ha un ingresso di CONVERSIONE INIZIALE, e possiede uscite tri-state. Contiene inoltre dei latch d'uscita e un segnale d'uscita FINE DELLA CONVERSIONE.

5) **Conversione dei Dati in Gradi Celsius**
Diagramma di Flusso:



USO DI UNA TABELLA DI CALIBRAZIONE

La conversione utilizza una tabella che contiene il valore d'ingresso più grande corrispondente a una data temperatura. Il programma esplora la tabella, alla ricerca di un valore maggiore od uguale al valore ricevuto dal convertitore. Il primo di tali valori che esso trova corrisponde alla temperatura richiesta; cioè, se il decimo ingresso è il primo valore maggiore od eguale al dato, la temperatura è di 10 gradi. Questo metodo basato sulla ricerca del valore è inefficiente ma adatto all'applicazione considerata.

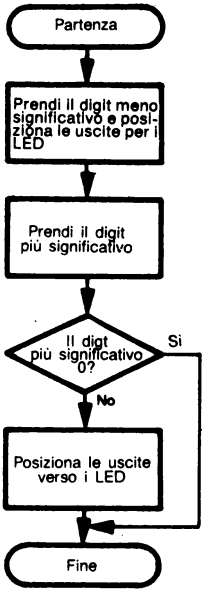
Notate che si deve mantenere il numero d'ingresso sotto forma decimale piuttosto che binaria. La sequenza di istruzioni «ADD A,1; DAA» conserva l'indice sotto forma di due cifre decimali invece che di un numero binario. Per esempio, il numero d'ingresso dopo il 9 (00001001 binario) sarà il 10 decimale (00010000 BCD) invece che il dieci binario (00001010). La ragione di tutto ciò consiste nel fatto che si è stabilito di visualizzare la temperatura con due cifre decimali, evitando in tal modo la conversione da binario a decimale.

La tabella potrebbe essere ricavata mediante calibrazione o mediante un'approssimazione matematica. Il metodo della calibrazione è semplice, dato che il termometro deve essere tarato in ogni caso. La tabella occupa una locazione di memoria per ciascun valore di temperatura da visualizzare.[1]

Per tarare il termometro, si devono per prima cosa regolare i potenziometri per estrarre l'esatto campo di variazione complessivo e quindi determinare i valori di uscita del convertitore corrispondenti alle temperature specifiche.

6) **Preparazione dei Dati per la Visualizzazione**

Diagramma di Flusso:



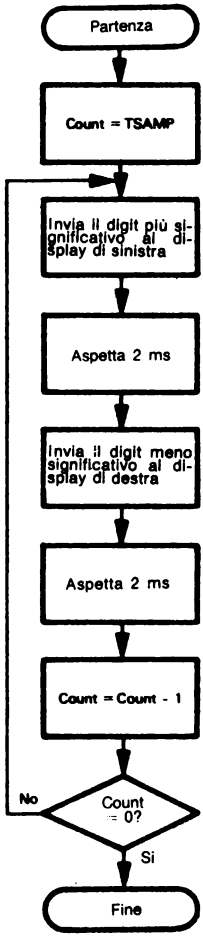
ELIMINAZIONE DI UNO ZERO DI GUIDA

La cifra meno significativa viene mascherata opportunamente. Si posiziona ad «1» il bit che accende i visualizzatori. Il risultato viene salvato nel Registro E.

La sola differenza per la cifra più significativa è che essa non compare se il suo valore è uguale a zero (cioè, i visualizzatori rappresentano «blank 7» invece di «07» per indicare 7°C). Tutto ciò porta semplicemente al mancato posizionamento ad «1» del bit che accende i visualizzatori nel caso la cifra sia zero. Il risultato viene salvato nel Registro D.

7) **Visualizzazione della temperatura per Sei Secondi**

Diagramma di Flusso:



Ogni visualizzatore viene attivato con una frequenza sufficiente a farlo apparire acceso di luce continua. Se TPULS fosse reso più lungo (diciamo 50 ms), il visualizzatore apparirebbe acceso di luce tremolante.
Il programma impiega un contatore a 16 bit per tenere il conteggio del tempo tra un campionamento e l'altro della temperatura. Lo Z80 possiede delle istruzioni per incrementare o decrementare coppie di registri a 16 bit o registri indice. Tuttavia, queste istruzioni non influenzano i «flag», così non c'è alcun modo per determinare direttamente l'istante in cui il contatore raggiunge lo zero.
Pertanto si esegue questa determinazione in maniera logica eseguendo l'operazione di OR tra gli otto bit più significativi e gli otto bit meno significativi del contatore. Se questo risultato è pari a zero, il contatore a 16 bit è arrivato a zero.

```
;
;NOME DEL PROGRAMMA: TERMOMETRO
;DATA DEL PROGRAMMA: 20/10/78
;PROGRAMMATORE: LANCE A. LEVENTHAL
;REQUISITI DELLA MEMORIA PROGRAMMI: 154 BYTE
;REQUISITI DI RAM: NESSUNA
;REQUISITI DI I/O: PORTA DI INGRESSO 1, PORTA DI USCITA 1 (1 PIO Z80)
;
;QUESTO PROGRAMMA È UN TERMOMETRO DIGITALE CHE ACCETTA INGRESSI
;   DA UN CONVERTITORE A/D COLLEGATO AD UN TERMISTORE, CONVERTE
;   L'INGRESSO IN GRADI CELSIUS E VISUALIZZA I RISULTATI SU DUE
;   DISPLAY LED A SETTE SEGMENTI
;
;CONVERTITORE A/D
;
;IL CONVERTITORE A/D È UN DISPOSITIVO ANALOGICO 7570J
;   (CONVERTITORE MONOLITICO) CHE PRODUCE UN'USCITA AD 8 BIT
;IL PROCESSO DI CONVERSIONE HA INIZIO DA UN IMPULSO SULLA
;   LINEA START CONVERSION (BIT 7 DELLA PORTA B DEL PIO)
;LA CONVERSIONE È PORTATA A TERMINE IN 50 MICROSECONDI E IL
;   DATO DIGITALE VIENE MESSO IN UN REGISTRO DI TIPO LATCH
;
;DISPLAY
;
;SI USANO DUE DISPLAY LED A SETTE SEGMENTI CON DECODIFICATORI
;   SEPARATI (7447 O 7448 IN FUNZIONE DEL TIPO DI DISPLAY)
;I DATI D'INGRESSO DEL DECODIFICATORE SONO COLLEGATI AI BIT
;   DA 0 A 3 DELLA PORTA B DEL PIO
;IL BIT 4 DELLA PORTA B DEL PIO È USATO PER ATTIVARE I
;   DISPLAY LED (IL BIT 4 È 1 PER INVIARE DATI AI LED)
;IL BIT 5 DELLA PORTA B DEL PIO È USATO PER SCEGLIERE QUALE
;   LED DEVE ESSERE USATO (IL BIT 5 È 1 SE SI DEVE USARE IL PRIMO DISPLAY,
;   O SE SI DEVE USARE L'ULTIMO DISPLAY)
;
;METODO
;
;PASSO 1 – INIZIALIZZAZIONE
;   LO STACK DI MEMORIA (USATO PER GLI INDIRIZZI DI RITORNO DELLE
;   SUBROUTINE VIENE INIZIALIZZATO
;PASSO 2 – IMPULSO SULLA LINEA START CONVERSION
;   LA LINEA START CONVERSION DEL CONVERTITORE A/D (BIT 7
;   DELLA PORTA B DEL PIO) RICEVE UN IMPULSO
;PASSO 3 – ATTESA DELL'ASSESTAMENTO DELL'USCITA DELL'A/D
;   UN'ATTESA DI UN MILLISECONDO PERMETTE
;   IL COMPLETAMENTO DELLA CONVERSIONE
;PASSO 4 – LETTURA DEL VALORE A/D, CONVERSIONE IN GRADI CELSIUS
;   SI USA UNA TABELLA PER LA CONVERSIONE CHE CONTIENE IL
;   MASSIMO VALORE DI INGRESSO PER OGNI LETTURA DI TEMPERATURA
;PASSO 5 – VISUALIZZAZIONE DELLA TEMPERATURA SU LED
;   LA TEMPERATURA È VISUALIZZATA SUI LED PER SEI SECONDI
;   PRIMA CHE SI EFFETTUI UN'ALTRA CONVERSIONE
;
;
;DEFINIZIONI DELLE VARIABILI DEL TERMOMETRO
;
;COSTANTI DEL SISTEMA DI MEMORIA
```

```
;
BEGIN   EQU     50H             ;INDIRIZZO DI PARTENZA
                                ;   DEL PROGRAMMA PRINCIPALE
LASTM   EQU     1000H           ;INDIRIZZO DI PARTENZA DELLO STACK DI RAM
;
;UNITÀ DI I/O E INDIRIZZI DEL PIO
;
PIODRA  EQU     0E0H            ;INGRESSI DEL PIO PER IL CONVERTITORE
PIOCRA  EQU     0E2H
PIODRB  EQU     0E1H            ;USCITA DEL PIO PER I DISPLAY
PIOCRB  EQU     0E3H
;
;DEFINIZIONI
;
LEDON   EQU     4               ;POSIZIONE DEI BIT PER MANDARE DATI AI LED
LEDSL   EQU     5               ;POSIZIONE DEL BIT PER SCEGLIERE
                                ;   IL PRIMO DISPLAY
MSCNT   EQU     0F9H            ;CONTEGGIO NECESSARIO PER DARE
                                ;   UN RITARDO DI 1 MS
STCON   EQU     10000000B       ;USCITA PER TENERE ALTA LA LINEA
                                ;   START CONVERSIONE
TPULS   EQU     2               ;DISPLAY PULSE LENGHT IN MS
TSAMP   EQU     1500            ;TSAMP È IL NUMERO DI VOLTE CHE I DISPLAY
                                ;   SONO FATTI PULSARE IN PERIODO
                                ;   DI CAMPIONAMENTO DELLA TEMPERATURA.
                                ;   LA LUNGHEZZA DI UN PERIODO DI
                                ;   CAMPIONAMENTO È COSÌ 2 TPULS TSAMP MS.
                                ;   IL FATTORE 2 TPULS È INTRODOTTO
                                ;   PER IL FATTO CHE OGNUNO DEI DUE DISPLAY
                                ;   È FATTO PULSARE PER TPULS MS.
;
;
;
        ORG     0
;
;ROUTINE DI RESET PER ARRIVARE AL PROGRAMMA TERMOMETRO
;
        JP      BEGIN           ;TROVA IL PROGRAMMA TERMOMETRO
;
;INIZIALIZZAZIONE DEL PROGRAMMA TERMOMETRO
;
        ORG     BEGIN
        LD      A,01001111B     ;RENDI LA PORTA A DEL PIO UN INGRESSO
        OUT     (PIOCRA),A
        LD      A,00001111B     ;RENDI LA PORTA B DEL PIO UN'USCITA
        OUT     (PIOCRB),A
        LD      SP,LASTM        ;METTI LO STACK ALLA FINE DELLA RAM
;
;FAI PULSARE LA LINEA START CONVERSION
;
START:  LD      A,STCON
        OUT     (PIODRB),A      ;METTI UN ALTO START CONVERSION
        SUB     A
        OUT     (PIODRB),A      ;METTI UN BASSO SU START CONVERSION
```

```
;
;RITARDO DI 1 MS PER LA CONVERSIONE
;
        LD      A,1             ;TEMPO DEL RITARDO DI CONVERSIONE IN MS
        CALL    DELAY           ;ATTESA DELLA CONVERSIONE
;
;LETTURA DEL DATO DIGITALE DAL CONVERTITORE
;
        IN      A,(PIODRA)      ;PRENDI IL DATO DAL CONVERTITORE A/D
;
;CONVERSIONE DEL DATO A/D IN 2 DIGIT BCD
;
        CALL    CONVR           ;CONVERTI IL DATO IN BCD
;
;PRENDI IL DIGIT MENO SIGNIFICATIVO
;
        LD      B,A             ;SALVA I DIGIT BCD
        AND     0FH             ;MASCHERA IL DIGIT LSD
        SET     LEDON,A         ;POSIZIONA LE USCITE DEI LED
        LD      E,A             ;SALVA IL DIGIT LSD NEL REGISTRO E
;
;PRENDI IL DIGIT PIÙ SIGNIFICATIVO, ELIMINA LO ZERO DI GUIDA
;
        LD      A,B             ;IMMAGAZZINA DI NUOVO I DIGIT BCD
        RRCA                    ;FAI SCORRERE IL DIGIT MSD
        RRCA
        RRCA
        RRCA
        AND     0FH             ;MASCHERA IL DIGIT MSD
        JR      Z,SVMSD         ;NON ACCENDERE IL DISPLAY SE IL VALORE È ZERO
        SET     LEDON,A         ;POSIZIONA LE USCITE DEI LED
        SET     LEDSL,A         ;SCEGLI IL PRIMO DISPLAY
SVMSD:  LD      D,A             ;SALVA IL DIGIT MSD NEL REGISTRO D
;
;FAI PULSARE I DISPLAY LED
;
        LD      C,PIODRB        ;PRENDI L'INDIRIZZO DELLA PORTA DI USCITA
        LD      HL,TSAMP        ;PRENDI IL CONTATORE DI IMPULSO A 16 BIT
DSPLY:  OUT     (C),D           ;USCITA DEL PRIMO DIGIT PER IL DISPLAY
        LD      A,TPULS         ;RITARDA LA LUNGHEZZA
                                ;   DELL'IMPULSO DEL DISPLAY
        CALL    DELAY
        OUT     (C),E           ;USCITA DELL'ULTIMO DIGIT PER IL DISPLAY
        LD      A,TPULS         ;RITARDA LA LUNGHEZZA
                                ;   DELL'IMPULSO DEL DISPLAY
        CALL    DELAY
        DEC     HL              ;CONTEGGIO ALLA ROVESCIA
                                ;   DEL CONTATORE A 16 BIT
        LD      A,H             ;RICORDATI CHE DEC HL
                                ;   NON POSIZIONA AD 1 IL FLAG Z
        OR      L
        JR      NZ,DSPLY        ;CONTINUA A FAR PULSARE I DISPLAY
        JP      START           ;VA NUOVAMENTE A CAMPIONARE
                                ;   LA TEMPERATURA
```

```
;SUBROUTINE DELAY CHE ATTENDE IL NUMERO
;   DI MILLISECONDI SPECIFICATO NEL REGISTRO A
;
DELAY:  EXX                     ;SALVA I REGISTRI DELL'UTILIZZATORE
DLY1:   LD      C,MSCNT         ;CARICA IL REGISTRO, COL RITARDO DI 1 MS
WTLP:   DEC     C               ;ATTENDI 1 MS
        JR      NZ,WTLP
        DEC     A               ;CONTEGGIO ALLA ROVESCIA DEL NUMERO DI MS
        JR      NZ,DLY1
        EXX                     ;IMMAGAZZINA DI NUOVO
                                ;   I REGISTRI DELL'UTILIZZATORE
        RET
;
;SUBROUTINE CONVR CHE CONVERTE L'INRESSO
;   DAL CONVERTITORE A/D IN GRADI CELSIUS USANDO UNA TABELLA.
;   IL DATO DI INGRESSO SI TROVA NELL'ACCUMULATORE.
;   IL RISULTATO SU 2 DIGIT BCD SI TROVA NELL'ACCUMULATORE
;
;REGISTRI USATI: A, B, C, H, L
;
;
CONVR:  LD      HL,DEGTB        ;PRENDI L'INDIRIZZO DI BASE
                                ;   DELLA TABELLA DI CONVERSIONE
        LD      B,A             ;SALVA L'INGRESSO A/D
        LD      C,0             ;PONI A ZERO I GRADI
CHVAL:  LD      A,(HL)          ;PRENDI L'INGRESSO PER LA TABELLA
        CP      B               ;C'È UN INGRESSO A/D SOTTO L'INGRESSO?
        LD      A,C             ;PRENDI IL VALORE IN GRADI CELSIUS
        RET     NC              ;SÌ, IL VALORE È STATO TROVATO
        ADD     A,1             ;NO, SOMMA 1 AI GRADI
        DAA                     ;TIENI I GRADI IN BCD
        LD      C,A
        INC     HL
        JR      CHVAL
;
;LA TABELLA DEGTB È STATA OTTENUTA MEDIANTE
;   CALIBRAZIONE CON UN RIFERIMENTO NOTO
;DEGTB CONTIENE IL PIÙ GRANDE VALORE CHE CORRISPONDE
;   ALLA LETTURA DI UNA PARTICOLARE TEMPERATURA
;   PER ESEMPIO IL PRIMO INGRESSO È 58
;   DECIMALE COSÌ UN VALORE DI INGRESSO PARI A 58
;   È IL PIÙ GRANDE VALORE CHE DÀ UNA LETTURA DI
;   TEMPERATURA PARI A ZERO – VALORI SOTTO ALLO ZERO
;   SONO VISUALIZZATI COME ZERO)
;
```

```
DEGTB:  DEFB 58
        DEFB 61
        DEFB 63
        DEFB 66
        DEFB 69.
        DEFB 71
        DEFB 74
        DEFB 77
        DEFB 80
        DEFB 84
        DEFB 87
        DEFB 90
        DEFB 93
        DEFB 97
        DEFB 101
        DEPB 104
        DEFB 108
        DEFB 112
        DEFB 116
        DEFB 120
        DEFB 124
        DEFB 128
        DEFB 132
        DEFB 136
        DEFB 141
        DEFB 145
        DEFB 149
        DEFB 154
        DEFB 158
        DEFB 163
        DEFB 167
        DEFB 172
        DEFB 177
        DEFB 181
        DEFB 186
        DEFB 191
        DEFB 195
        DEFB 200
        DEFB 204
        DEFB 209
        DEFB 214
        DEFB 218
        DEFB 223
        DEFB 227
        DEFB 232
        DEFB 236
        DEFB 241
        DEFB 245
        DEFB 249
        DEFB 253
        DEFB 255
        END
```

BIBLIOGRAFIA

1. Un metodo che impiega assai meno memoria viene descritto in T. A. Seim, "Numerical Interpolation for Microprocessor-based Systems", Computer Design, Febbraio 1978, pp. 111-116.

Vedere inoltre:

2. Auslander, D. M. et al., "Direct Digital Process Control: Practice and Algorithms for Microprocessor Applications", Proceedings of the IEEE, Febbraio 1978, pp. 199-208.

3. Bernstein, N., "What to Look for in Analog Input/Output Boards", Electronics, 19 gennaio, 1978, pp. 113-119.

4. Bibbero, R. J., Microprocessors in Instruments and Control, Wiley, New York, 1977.

5. Burton, D. P. and A. L. Dexter, Microprocessor Systems Handbook. Analog Devices, Inc. P.O. Bpx 796, Norwood, MA. 02062, 1977.

6. Finkel, J., Computer-Aided Experimentation, Wiley, New York, 1975.

7. Garret, P. H., Analog Systems for Microprocessors and Minicomputers, Reston Publishing Co., Reston, VA., 1978.

8. Hnatek, E. R., A User's Handbook of D/A and A/D Converters, Wiley, New York, 1976.

9. Mrozowski, A., "Analog Output Chips Shrink A-D Conversion Software", Electronics, 23 Giugno, 1977, pp. 130-133.

10. The Optoelectronics Data Book. Texas Instruments, Inc., P.O. Box 5012, Dallas, TX., 1978.

11. The Optoelectronic Designer's Catalog, Hewlett-Packard Inc., 1820 Embarcadero Road, Palo Alto, CA. 94303, 1978.

12. Peatman, J. B., Microcomputer-based Design, McGraw-Hill, New York, 1977.

13. Rony, P. R. et al., "Microcomputer Interfacing: Sample and Hold Devices", Computer Design, Dicembre 1977, pp. 106-108.

14. Sheingold, D. H. ed Analog-Digital Conversion Notes, Analog Devices, Inc., P.O. Box 796, Norwood, MA. 02062, 1977.

L'Autore

Lance A. Leventhal è un collaboratore di Emulative Systems Company, una ditta di consulenza con sede a San Diego specializzata in microelaboratori e microprogrammatori. Egli opera come Techinical Editor per la Society for Computer Simulation e come Contributing Editor per Digital Design. È relatore nazionale per i microelaboratori per la IEEE, autore di cinque libri e oltre quaranta articoli sui microelaboratori e anche regolare collaboratore per pubblicazioni quali Simulation, Digital Design e Kilobaud.

Precedentemente il Dr. Leventhal ha collaborato con Linkabit Corporation, Intelcom Rad Tech, Naval Electronics Laboratory Center ed Harry Diamond Laboratories. Ha ottenuto un titolo B.A. alla Washington University di St. Louis nel Missouri e titoli M.S. e Ph. D. presso l'University of California di San Diego. È membro SCS, ACM e IEEE.

L. 29.500

Cod. 326 P

36

Z 80 PROGRAMMAZIONE IN LINGUAGGIO ASSEMBLY

Lance A. Leventhal

GRUPPO EDITORIALE JACKSON